ACKNOWLEDGMENTS

This thesis is the result of six months of hard work. Although I attempted to do most of the hard work myself, I am certainly not the only one who put effort into this thesis. I especially would like to thank my supervisors Jan van den Berg and Erik van Mulligen, who have read and re-read the texts I produced so many times they must know most of it by heart.

Additionally, I would like to thank the other reviewers who, with their many comments, contributed to the quality of this thesis. Thank you, Jan Kors and Bob Schijvenaars. I would also like to thank David Brée for checking my English and the consistency of the thesis.

I would like a few other people whose contributions are more indirect. My girlfriend Judith has been an important source of support, as have my parents. I would also like to thank Cobus van Wyk for providing distractions when they were needed.

Christiaan van der Eijk
Rotterdam, 15 October 2001

C O N T E N T S

1 INTRODUCTION

1.1 Motivation

Communication is essential for scientists. Communication of ideas, discoveries and results plays a crucial role in scientific progress. Scientists need to build on the work of their predecessors and can only do so if these predecessors have transmitted their ideas. Clearly, in the scientific world, communication takes on different forms. The most basic is the discussion: scientists meeting at coffee machines, at meetings and conferences discuss their research. Other communication takes place in written form. Some modes of written communication are informal, such as e-mails; others are more formal, such as reports, proceedings, journal articles and books.

Transmission of knowledge is the important goal of scientific communications. However, not all knowledge has the same status. Knowledge can be unconfirmed and under debate, or it can be viewed as a fact, or somewhere in between these two extremes. Articles in journals or books contain knowledge that has been reviewed by peers. When many scientists have accepted knowledge because it resists falsification, it moves from hypothesis to the realm of accepted knowledge.

Being able to find one's way in these communications can contribute greatly to a scientist's success. Filtering out the most relevant information in the mass of spoken and written information is extremely difficult because of the sheer amount of scientific output both accepted and hypothetical, and because of the heterogeneity of scientific communication.

An important source of information both of one's own field and that of others is scientific literature. But even keeping up with all that goes on in one's own community takes up large amounts of time. The number of medical journals, for example, has doubled every 19 years since 1870 (1, 2). To keep up with the 10 leading journals in internal medicine alone, it would be necessary for a clinician to read as many as 200 articles and 70 editorials per month (3).

Moreover, because of the increasingly narrow focus of documents, the answer to scientific questions will often be found not by reading a single document, but by combining information in several documents from different research fields. Someone with a query will require a system to return several documents that together form the result searched for. Independent articles contain part of the answer; only by bringing them together, the user has sufficient information to formulate an answer. Researchers can and have made new discoveries without doing any experiments themselves, but by merely combining results from one article with those of another. A scientist who finds a relation between the entities *a* and *b* in one article, and between *b* and *c* in another, will be able to hypothesize that *a* and *c* are related, even if no other literature supports this notion. Comparing documents for partial overlap is, unfortunately, a time-consuming task, because of the sheer number of combinations of articles. Computer assisted search could provide part of the solution.

A prime example of combining knowledge in existing literature is the work of Don Swanson (4). By sifting through medical literature he discovered that fish oils have a positive effect on patients suffering from Raynaud’s disease. One set of articles reported a negative correlation between the use of fish oils and blood platelets aggregation and a different set of articles showed platelet aggregation is a characteristic of Raynaud’s disease. Swanson postulated on the basis of the articles that fish oil might treat this disease. Clinical trials later showed that fish oils do indeed treat Raynaud’s disease, a discovery based wholly on underused knowledge in literature.

Discoveries across disciplines can lead to interesting result as Swanson shows. However, acquiring knowledge from different fields is especially difficult, because of increasing specialization in every branch of science (5). Scientific communities develop results independently of each other, often unaware of related research carried out in other communities. Serendipity, the faculty of making happy and unexpected discoveries by accident, is suppressed because researchers come across fewer insights from other fields.

In this thesis, an attempt will be made to develop a system that aids scientists in their search for knowledge in literature. For that purpose, knowledge representation, knowledge discovery and information retrieval will be discussed. The last and larger part of my thesis will be dedicated to the design and evaluation of a system for *knowledge discovery in literature*.

This research focuses on medical literature for several reasons. The most important is that I will make use of other systems that work best for medical literature. In addition, medical literature has already been the subject of much research by knowledge discovery scientists, so I can build on knowledge generated by others, as is fitting of this field. However, it is quite easy to generalize the central ideas of this thesis to knowledge discovery in other disciplines.

1.2 Background

The goal of this thesis is to develop a system that will help researchers to relate concepts which were previously unconnected in literature. In order to do so terminology is needed. Here only the terminology required for setting up the structure of the thesis is introduced. It will be discussed in greater detail in the next chapters.

Central to this thesis are the notions of *concept* and that of *relation between concepts.* Both need a careful definition. A concept, in the way it is used in this thesis, refers to a certain *object*, *idea* or *activity.* Terms are natural language expressions used to denote a concept. The concepts itself exists outside of language. Terms or words are parts of a language and concepts are a part of our understanding of reality. Terms are mere labels or names of the entity. This model is chosen for practical, and not for philosophical reasons.

Different terms may be used to label the same concept and such terms are then called synonyms or translations. The English term ‘congenital abnormality’ labels a certain concept, and the Dutch equivalent ‘aangeboren afwijking’ labels this same concept. To complicate things, one term can also label several different concepts. For example the word ‘cold’ can be used in the meaning of low temperature, but also of a common ‘cold’. In this case one term, ‘cold’, labels two different concepts (6). Concepts are, in short, the entities we label with our words. The reason concepts will be the major concern in this thesis is that we are trying to tie together literature from different sources that often use alternative terms for the same thing. Philosophers, linguists and cognitive scientists have debated the definition of concepts and their relations to reality and some of this discussion can be found in chapter 2.

Concepts are central to this thesis, but because concepts exist outside of language some sort of label is needed to be able to discuss them. This can be a number or a natural language term. For example in this thesis the concept 'malaria' might be mentioned. By saying 'the concept malaria' and not 'the term malaria' the concept denoted by the term malaria is meant and not just it's label.

Concepts can be related in many ways. A relation exists between a pair of concepts, possibly with certain attributes. A 'transport' relation for example connects 'mosquitoes' to 'malaria'. 'Leg' is connected by an 'is-a' relation to 'limb'.

An article, especially a scientific one, can be seen as a set of statements about concepts and the way they are interrelated. From this perspective, every article represents a *network* of concepts where a network is a set of concepts and a set of links between concepts. The more structured the article, the easier it is to distill a network from the sentences. The networks of several non-contradictory articles together form a larger network of concepts. Scientific knowledge in a certain field can be represented in such a network. An example of this representation of concepts from computer science is given in Fig. 1.1. Although it is by no means an accurate representation of the field, it illustrates the idea of knowledge network. The graph consists of concepts with labels like 'Data mining' and relations between concepts.

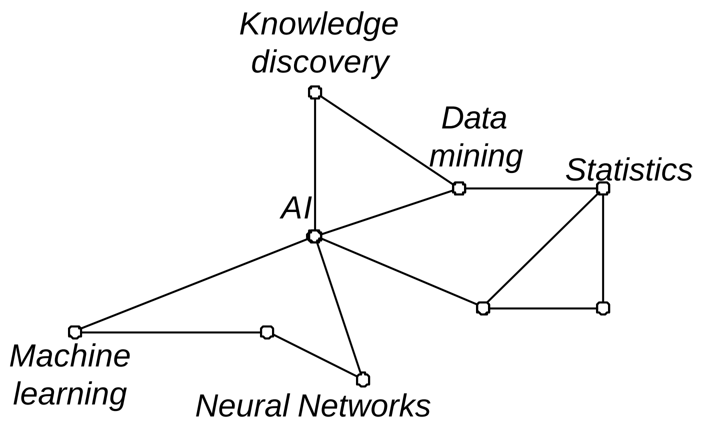


Figure 1.1 A knowledge network

In a network of knowledge one can walk from one concept to the next via relations. A walk or *path* is an ordered set of concepts where each two consecutive concepts are related. An example of a path of this type is given by Swanson's hypothesis: fish oil –*lowers*- platelet aggregation –*characteristic of*- Raynaud's disease. These relations can differ widely in nature and type and as a result the same is true for paths.

The system to be developed must help the user find paths like Swanson's. To do so the system must be able to generate a network of knowledge for a certain discipline and help the user navigate through this knowledge network. This involves finding the relevant concepts and how they are related in a body of literature.

There are several ways of looking at literature. One way is to take each article individually and try to reconstruct the network of knowledge it represents. When the system understands the network in that particular article, it integrates the article's small network into the greater network of existing knowledge. This means that certain concepts that were introduced for the first time in the article are added to the greater network and new links are placed between concepts. An illustration of the merging process is found Fig. 1.2.

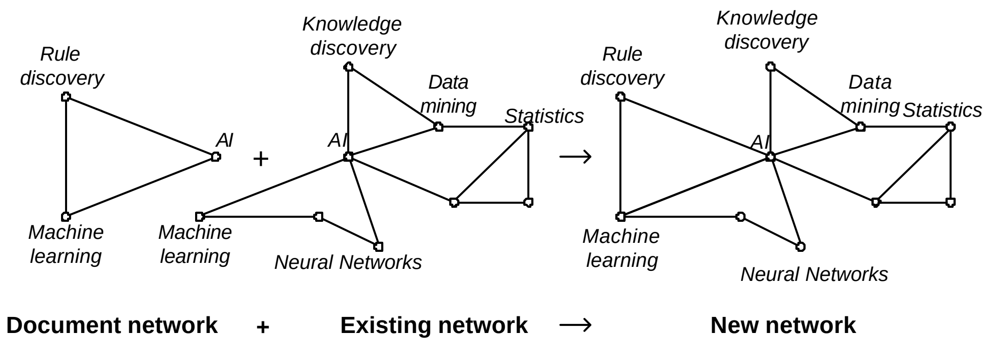


Figure 1.2 Merging networks

The second approach looks at knowledge from a higher aggregation level. In this perspective each article contains concepts. In a set of several articles, concepts will be distributed unevenly over the set, with related concepts often appearing together in a sentence, paragraph, chapter or document. Statistical analysis of articles can reveal these patterns of distribution of concepts in the article set. Some concepts, for example, occur frequently together in one article, while others never appear in the same piece of literature. A graph is constructed using concepts as nodes and connecting these nodes on the basis of statistical information concerning the

distribution of the concepts. How the relations are to be labeled will be discussed in chapters 2,3 and 6. It is this statistical approach that will be taken in this thesis.
The focus will be on the statistics relating concepts instead of words or terms. For this reason articles cannot be used directly, but are first preprocessed using an indexing system. The result of preprocessing an article is a list of relevance-weighted concepts found in that article, where the relevance weights indicate the relevance of the concept within the article.

1.3 Research Topics

The terminology introduced in the last few paragraphs makes a formulation of the central goal of this thesis possible. The goal is:

to find knowledge paths in scientific literature.

The knowledge paths to be found are series of concepts that are serially related. The ground for supposing a relation is in scientific literature. The system to be assessed here is aimed at scientists who suspect a relation between two concepts, but who cannot find this relation in single documents. These scientists will expect the system to be as fast as Information Retrieval systems, which means that processing times must be low.

1.4 Methodology

The goal of this thesis is to develop and evaluate a system for finding knowledge pathways in literature. In this section the steps taken towards achieving that goal are introduced.
The thesis is part of an on-going effort to develop ways to translate knowledge in natural language documents, best handled by humans, to symbolic information, manageable for computers.
To set the work done in this thesis in perspective, first the nature of the relations between knowledge, language and the objects of study are discussed. This discussion will be briefly theoretical, but has practical implications later on. It is followed by a review of the ways knowledge is used in current information systems. These systems

use different approaches to deal with information in natural language documents. The insights in how documents can be dealt with will give rise to new requirements.

The ideas used for handling texts in information systems will be used to develop a representation of knowledge in literature. This first representation will be a graph, called a co-occurrence graph. The choices made in the development of the co-occurrence graph will be partly based on theoretical and partly on practical considerations.

The use of co-occurrence graphs is problematic because of their size and certain reflections on the nature of the graph. To deal with these problems, the Associative Conceptual Space (ACS) is introduced. The workings of the ACS as it is described in M. Schuemie's thesis, are explained (7). It will be explained how the ACS can help deal with the shortcomings of the co-occurrence graph in this research.

The ACS will be adapted to suit the needs for this thesis. The adaptation is called the Conceptual Associative Spatial Graph (CASG). Improvements are suggested on theoretical grounds. The CASG is the system proposed to reach the goal.

The next question is one of evaluation. Does the CASG deliver what we want it to do? This question will be answered first on artificially generated data and later with real data from a set of articles. After these experiments conclusions will be drawn. Because the CASG leaves ample space for improvement suggestions will be given on how to improve it.

1.5 Structure

The thesis consists of 6 chapters.

Chapter 1 introduces the goal and the context of the thesis. It describes how the thesis is set up.

Chapter 2 explains how knowledge is viewed in this thesis and how others have used it.

Chapter 3 presents how knowledge is represented and used in this thesis. The limitations of this representation are discussed.

Chapter 4 introduces an addition to the representation of chapter 3. In this chapter the implementation of the new combined system is introduced.

Chapter 5 is an evaluation of the system developed in chapter 4.

Chapter 6 brings the thesis to an end with a final conclusion and suggestion for further research.

2 Themes in knowledge discovery

The aim of this research is to assist a reader to find concept relations in scientific literature. As was already mentioned in the introduction, this thesis stands in a longer line of research. This chapter will present different angles that have been taken in previous work and discuss the advantages and disadvantages of each. The chapter starts with a summary of the debate on knowledge representations (2.1). We then discuss two ways that have been used to query for knowledge in documents. The first, Information Retrieval (IR), is discussed in 2.2. The second approach, Knowledge Discovery (KD), is looked at in 2.3. At the end of the chapter the results of the discussion are used to refine of the research goal (2.4).

2.1 Knowledge representation

In chapter 1 the notions of concept, relation and network were introduced and the philosophical complications that they entail were mentioned. In this section these concepts will receive more attention. Although their treatment will be brief, some form of introduction is needed for better understanding of concepts and meaning. First concepts will be discussed and then their relations and networks. It will become clear that several, sometimes contradictory, definitions exist in the literature.
Concepts have been a matter of philosophical debate for centuries. The notion of concept arises from the problematic relation between words, thoughts and things (8). The German philosopher and logician Frege (1848-1925) was a significant contributor

to the thinking on meaning. Ogden and Richardson later represented his ideas in a *meaning triangle*.

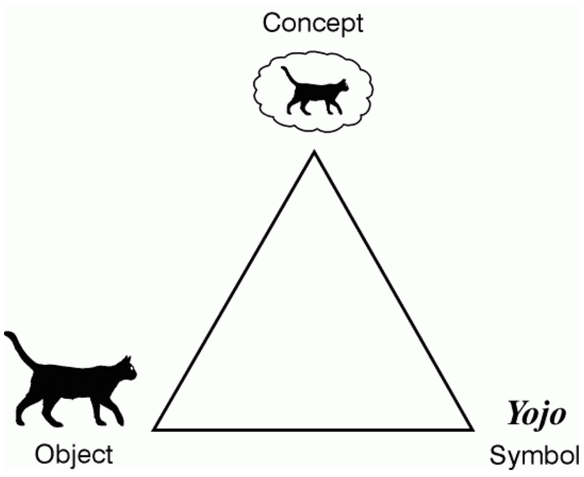


Figure 2.1 The Ogden-Richardson meaning triangle (9)

The meaning triangle is made up of three entities. There is an *object*, a real cat. There is a *concept*, which is the representation formed in the brain from the rays of light reflected of the object and its surroundings. Thirdly there is the *symbol* “Yoyo” which is a label or name of the object.

The relation between the symbol and the object itself is indirect and mediated by the interpreter. On hearing, seeing or reading the symbol the interpreter develops thoughts (the concept) and these thoughts are then linked to the thing in the world. Only through an interpreter a link can be made between the symbol and the object.

There is a problem with this model: it assumes that a one-on-one relation can exist between a word and a thought and between a thought and an object. Today many people recognize that such a model is an oversimplification of complex referential issues in communication. The examples in Chapter 1 support this idea. Multiple words can refer to one object and one word can refer to several distinct objects.

In the medical sciences, as in many others, unambiguous communication is essential. This brings a strong desire to have one-on-one relationships between thoughts, words and objects. This desire has given rise to the Unified Medical Language System (UMLS), which codifies equivalencies of terms used in the medical sciences (8). Several *thesauri* or structured vocabularies from the medical world are linked together in the UMLS, which defines for terms from different thesauri to which other terms they are equivalent. Thesauri in general and UMLS are discussed in greater detail in 2.2.2.

2.2 Information Retrieval

The explosion of the number of electronically available documents since the nineteen sixties has made research into computer processing of text worthwhile. This explosion of the amount of electronically available text has also taken place in Medicine. With the arrival of Internet, this rapid growth has only gained momentum. Much literature has become better accessible. An example of medical literature is PubMed, an archive of 11 million abstracts from articles in life science journals (10). These abstracts are freely accessible via the Internet. Literature, patient records and other text in electronic from have aroused the interest of information scientists.

Dealing with these documents has been approached in several ways. Research in the IR field has focused on retrieval of documents. This involves ways of understanding documents and understanding the users request for information. Because advanced retrieval systems attempt to use knowledge in documents to find the most relevant documents, there is a certain overlap between IR and KD. The key difference is that IR systems are meant to select documents, while KD systems present knowledge not just in document form, but in other representations as well. The IR field will be treated more in depth in this section.

KD is directed at finding ways, not just of providing the user with documents as they were entered, but processing the information so that information can be represented in different ways.

Although our aim is to go beyond basic retrieval, Information Retrieval is of interest. The reason is that many IR search systems use advanced techniques both to represent queries and documents in useful ways for better results.

2.2.1 Keyword based systems

Traditional Boolean IR systems require the user to enter keywords. The search engine then retrieves all documents containing that set of keywords from the collection of documents. This simple principle has disadvantages. A first drawback is the inability to deal with synonyms. Searching for one term will not give you documents containing an alternative name of the same concept. A second problem is defining the right keywords. Entering a too general set of few keywords will result in a very large and unmanageable set of documents, entering too specific keywords can leave the user empty-handed. Boolean systems restrict the search area, by only looking at a certain part of the documents. Adding keywords or changing an OR to an AND operator further decreases the search space.

These two problems are the consequence of a low level of abstraction. The user is usually not interested in documents containing certain words, but in documents about certain concepts. Abstracting from the word-level to the level of concepts would eliminate part of this problem. This step requires some form of concept recognition in text.

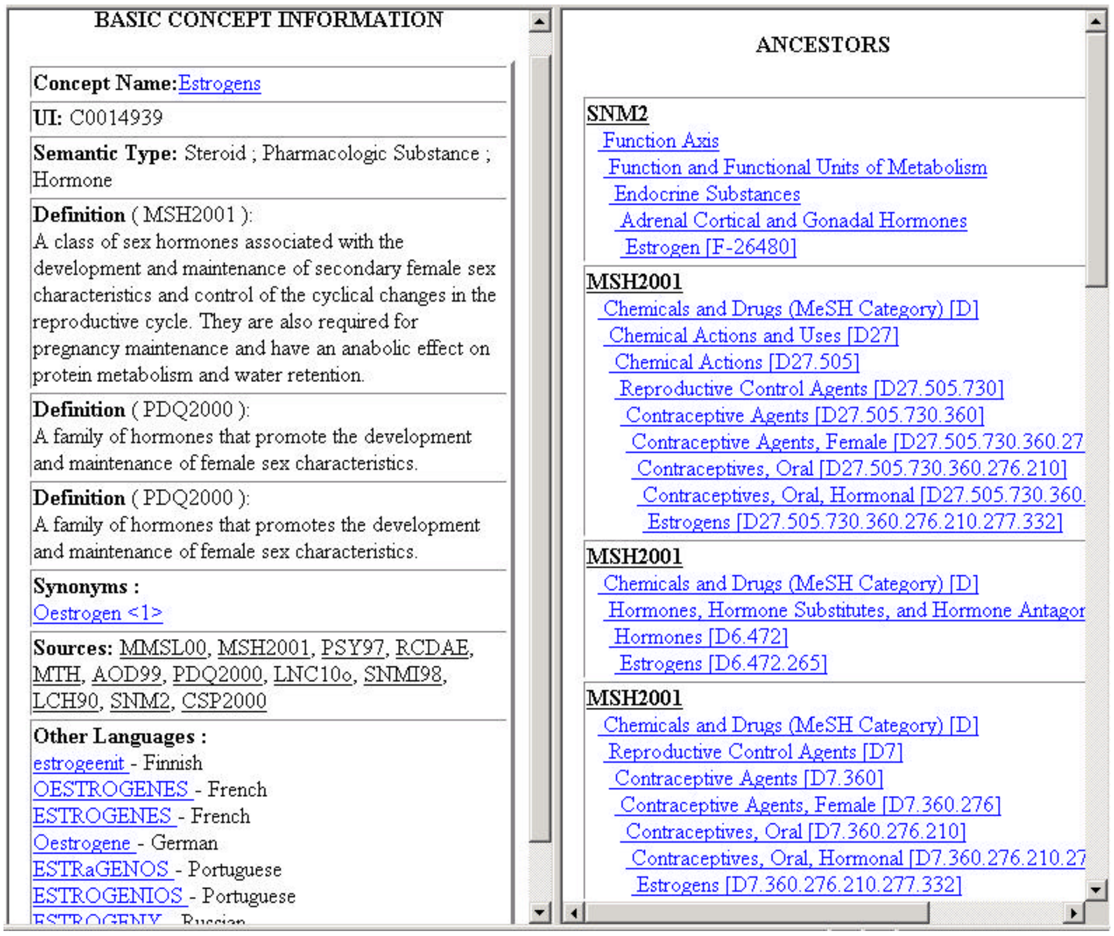
2.2.2 Thesauri and ontologies

To deal with the problem of synonyms and ambiguities, thesauri and *ontologies* have been developed. Both are codifications of equivalencies of terms that aim to clarify communication. In this subsection first the distinction between thesauri and ontologies is explained, after which the UMLS ontology receives more attention.

A thesaurus is an extension of a dictionary. A dictionary typically contains a list of numbered concepts, with the terms used to refer to these concepts. A thesaurus adds hierarchical relations to this list. These relations are of the “is-a” type. Thesaurus systems translate a query and documents from normal words to concept numbers. Mapping the, possibly many, synonyms to a concept number for both query and documents ensures that the match between these is made whenever there is a conceptual correspondence.

The next step up the semantic ladder are ontologies. Ontologies add semantics to thesauri. Concepts can be related to each other by means of predefined relations. An example of such an ontology is the UMLS (6). This ontology is the subject of the remainder of this section.

The UMLS was developed by the National Library of Medicine (NLM) to facilitate the use of computers for information usage in medicine. It was compiled from several previously existing thesauri. The 1999 edition of the UMLS holds more than 700,000 different concepts. Concepts in the UMLS have a semantic categories assigned to them, hierarchical relations, definitions, synonyms and terms in different languages as is illustrated in Fig. 2.1. The addition of semantic categories and different semantic relations are what makes the UMLS an ontology and not merely a thesaurus. This semantic add-on is called the Semantic Network. The part used in this thesis is the Metathesaurus, which is a thesaurus.



BASIC CONCEPT INFORMATION

Concept Name: Estrogens

UI: C0014939

Semantic Type: Steroid ; Pharmacologic Substance ; Hormone

Definition (MSH2001):
A class of sex hormones associated with the development and maintenance of secondary female sex characteristics and control of the cyclical changes in the reproductive cycle. They are also required for pregnancy maintenance and have an anabolic effect on protein metabolism and water retention.

Definition (PDQ2000):
A family of hormones that promote the development and maintenance of female sex characteristics.

Definition (PDQ2000):
A family of hormones that promotes the development and maintenance of female sex characteristics.

Synonyms :
Oestrogen <1>

Sources: MMSL00, MSH2001, PSY97, RCDAE, MTH, AOD99, PDQ2000, LNC10o, SNMI98, LCH90, SNM2, CSP2000

Other Languages :
estrogeenit - Finnish
OESTROGENES - French
ESTROGENES - French
Oestrogene - German
ESTRaGENOS - Portuguese
ESTROGENIOS - Portuguese
ESTROGENY - Russian

ANCESTORS

SNM2
Function Axis
Function and Functional Units of Metabolism
Endocrine Substances
Adrenal Cortical and Gonadal Hormones
Estrogen [F-26480]

MSH2001
Chemicals and Drugs (MeSH Category) [D]
Chemical Actions and Uses [D27]
Chemical Actions [D27.505]
Reproductive Control Agents [D27.505.730]
Contraceptive Agents [D27.505.730.360]
Contraceptive Agents, Female [D27.505.730.360.27
Contraceptives, Oral [D27.505.730.360.276.210]
Contraceptives, Oral, Hormonal [D27.505.730.360.
Estrogens [D27.505.730.360.276.210.277.332]

MSH2001
Chemicals and Drugs (MeSH Category) [D]
Hormones, Hormone Substitutes, and Hormone Antagor
Hormones [D6.472]
Estrogens [D6.472.265]

MSH2001
Chemicals and Drugs (MeSH Category) [D]
Reproductive Control Agents [D7]
Contraceptive Agents [D7.360]
Contraceptive Agents, Female [D7.360.276]
Contraceptives, Oral [D7.360.276.210]
Contraceptives, Oral, Hormonal [D7.360.276.210.27
Estrogens [D7.360.276.210.277.332]

Figure 2.2 UMLS entry of concept C0014939 (Estrogens)(6)

Figure 2.2 shows the output of the Metathesaurus of the UMLS for estrogens. This concept has a unique number, C0014939, a preferred name, 'estrogens', and a synonym 'oestrogen'. It is an example of different semantic types: 'steroid', 'pharmacologic substance' and 'hormone'. It has different names in different languages and it is defined in several thesauri. The ancestors as they are defined in the different thesauri UMLS is based on, are listed in the right window. The semantic relations with other concepts are not listed here.

Committees at the NLM maintain the ontology. These committees will have to reach an agreement on the meaning of a concept. The aim of this activity to include as many as possible and to make them as informative as possible. A very broadly defined term is not very informative, because it allows many different interpretations. For very specialized terms, however, disagreement often remains. Because consensus has to exist in the committee, the concepts defined are neither very general nor ve ry specific.

2.2.3 Collexis indexing and search

The software company Collexis has developed a system that can use the UMLS ontology for search. This is done in a way to solve both the problem of synonyms and of choosing the right set of keywords. The Collexis system creates a profile, called a *conceptual fingerprint*, for each document in its collection. This fingerprint is a set of concepts and a *relevance score* for each concept. The score indicates the relevance of the concept for the representation of the document and the 'confidence' in its identification. Figure 2.3 is the fingerprint of this thesis, created using a thesaurus for Computer Science, called Excerpta Informatica. The concepts in the fingerprints used in this thesis are ordered from high to low relevance.

l-system	1.0000	user requirements	0.0832
intelligent tutoring system	0.9959	nouns	0.0797
literature	0.6226	algorithm	0.0745
mis science	0.3953	binary search tree	0.0676
two-d	0.2739	philosophy	0.0640
network	0.2305	data description	0.0636
communication	0.2253	statistics	0.0591
semantics	0.2183	merging	0.0591
thesaurus	0.2068	hierarchical model	0.0543
user	0.2035	evaluation	0.0527
data representation	0.1996	parsing	0.0513
ups	0.1992	interpreter	0.0498
call	0.1940	internet	0.0485
information	0.1792	subject indexing	0.0474
verbs	0.1595	standardisation	0.0454
medicine	0.1329	case	0.0445
journal	0.1283	group decision making	0.0433
intermediary	0.1181	record identification	0.0412
vectors	0.1063	book	0.0387
lexicography	0.1040	data collection	0.0356
data base	0.0886	gas distribution	0.0341
introduction	0.0886	move	0.0295

drawing	0.0295	implementation	0.0257
newspaper	0.0295	operator	0.0249
mixed hardware	0.0295	mapping	0.0249
sporting competition	0.0295	selection	0.0237
text processing	0.0291	three-d	0.0231
people	0.0278	searching	0.0223
join	0.0278	conversion	0.0219
correspondence	0.0278	information retrieval	0.0196
proceedings	0.0278	data abstraction	0.0178
user behaviour	0.0278	library	0.0144
abstracting	0.0278	electronics	0.0124
trend	0.0266	man machine	0.0119
archive	0.0266	chromosome analysis	0.0112
knowledge representation	0.0257	software	0.0085
value added network	0.0257		

Figure 2.3 Fingerprint of this thesis

An algorithm creates these fingerprints. This algorithm removes stop words like ‘is’, ‘of’, etcetera. The result of this step is a document with the most common verbs and prepositions removed. In the next step verbs and nouns are normalized to a standard form. Verbs are changed to their present tense singular and nouns are changed to their plural number, if not already in that form. Using the list of names and synonyms from the thesaurus the program identifies the concepts present in the resulting document. The algorithm assigns these concepts of to the fingerprint, preferring *specific* concepts over general concepts. A term is more specific than another if it is positioned lower in the hierarchy of the ontology. Different kinds of malaria, like malaria Vivax and malaria Falciparum are more specific than malaria in general.

A relevance score is calculated for each concept on the basis of the frequency of occurrence of its various terms in the document, the number of its synonyms found, the number of different individual words from which the terms are made up that are found, and the specificity of the concept. The resulting relevance score is an indication of the importance of the concept in the document and the confidence of its identification.

The result is a list as the one in Fig. 2.3 of all concepts relevant for a single document and their score. This list is represented as an *n*-dimensional vector where *n* is the number of concepts in the ontology. When the user enters a query the same algorithm is used for representing the query as an *n*-dimensional vector. The best matching document is the document whose vector is closest to the query vector. This approach solves both the synonym problem and the solution space problem. The synonyms are dealt with by the conversion to ontology entries and the solution space is no longer searched by gradually restricting it like with Boolean systems, but through a linear search of the whole space.

2.3 Knowledge Discovery

Knowledge discovery in medical literature has been the subject of research for some time. Several articles focus on *Natural Language Processing* (NLP) techniques to extract useful information from articles and abstracts. A second line of research is directed at finding clusters and relations of terms or concepts in medical text through statistics.

2.3.1 Natural Language Processing

Systems that use NLP methods try to ‘read’ a document and distill from the text concepts and their relations. This means that every sentence is mapped onto a tree, called a parse tree, which represents the structure of the sentence, and all words and groups of words in the sentence are labeled as nouns, verbs, etc. NLP systems, if and when they work, make use of a great part of the information stored in a text. NLP systems can translate a human-language article in a structured format a computer can work with.

Parsing unstructured text is, however, a computationally expensive process. That is the reason many systems do not try to translate all of the information in an article to a structured format. Instead many authors of articles on NLP restrict themselves to certain specific concepts or relations. This limited form of NLP is often called *Information Extraction* (IE). A good example of this can be found in (11). In this article a system is proposed that finds five types of relations involving proteins, e.g. between proteins and diseases. The systems parses medical abstracts and uses a Bayesian classifier to classify parts of sentences as positive examples or negative examples of one or other of these five types of relations. Another example is found in

(12) were partial parse trees are used instead of full trees. Sekimizu et al. (13) use partial parsing and the UMLS metathesaurus to identify genes and gene products in literature. They then use a list of frequent verbs to classify relations between genes and gene products. (14) works in a similar vein, but looks for genes and drugs and their relations.

The main problem, especially when it comes to identifying relations between concepts, is that human language is often complex and full of ambiguities, which are easy for humans to resolve, but difficult to model. Many sentences take the following form: 'By contrast, activated H-ras, which acts downstream of src, failed to induce resistance to either of these drugs.' (14) For NLP systems it is extremely difficult to find out what drugs are referred to by *these* drugs. It would need to identify the drugs in previous sentences and decide which ones are the drugs implied in the sentence. Less hard, but still sufficiently difficult to make the identification of relations impractical, is dealing with negations and verbs in the passive mood.

NLP systems are still too slow and inaccurate to be implemented fully, while the faster IE systems have a rather limited scope. Given their potentially exhaustive use of textual information these approaches could become more rewarding in the future.

2.3.2 Statistical Systems

Statistical systems look at texts at a different level of detail. Many statistical systems use co-occurrence of concepts in a document as a measure. The central assumption in this approach is that frequently co-occurring concepts must be related.

For a set of texts they count how frequently words occur together in the same sentence, paragraph, article, or arbitrarily sized window. The frequency or relative frequency of their co-occurrence serves as an indication of the 'relatedness' of the concepts.

It is in this area that Swanson's work can be placed (4). He proposes a system in which the user selects a term, e.g. Raynaud's disease for a MedLine search. The system retrieves a set of articles S. It then generates all title words from *S*, with exception of non-informative or too general words. This set of words is used for a query that results in a new set of articles *T*. All title words from *T* are placed in a word set *R*, with certain restrictions. Between the seed term (Raynaud's disease) and the new terms (*R*) lies a set of *intermediary terms* (the title word set *S*). *R* is ordered in order of frequency of occurrence in the set. Thus, potentially interesting concepts are generated.

Although the idea behind Swanson's system is very simple, the results are impressive. At least two discoveries from a textual database have indeed sparked interest in the medical community. The system has its weaknesses, however. It only takes titles into account and makes no use of information in the article or abstract. The system can deal neither with synonyms nor with aliases. Another problem is that it requires quite some effort from the user, who first has to select a seed set of articles and then all potentially interesting words among the intermediary terms. The last issue is that it allows only one intermediary term between the target and the source term. Marc Weeber has done work similar to that of Swanson's (15). Using MedLine abstracts he tried to find adverse drug reactions of medicines that in other contexts could be used to treat diseases. To identify the side-effect related words he uses the UMLS thesaurus.

Stapley and Benoit find relations between pairs of genes on the basis of their co-occurrence in a selection of Medline abstracts (16). They simply use the reciprocal of the Dice coefficient between two genes as a similarity measure.

The reciprocal of the coefficient of gene *i* and gene *j* is defined as:

$$d_{i,j} = \frac{|i|+|j|}{|i \cap j|} \text{, and } 2 \le d_{i,j} \le \infty \qquad (2.1)$$

where $|i|$ is the number of articles in which gene *i* occurs and $|i n j|$ the number of articles in which *i* and *j* both occur.

The findings are then represented in a graph where the genes are the nodes. An edge exists between gene *i* and gene *j* if and only if $d_{i,j}$ is greater then a certain threshold. This approach seems somewhat modest in scope because of the restriction to gene names. Veeling et al. have used co-occurrence graphs for grouping concepts from a database of short newspaper articles (17). These groups are used to aid the user in a document retrieval task.

Schuemie uses a different approach (7). He does not construct a co-occurrence graph, but rather an *n*-dimensional space. Concepts are initially randomly placed in a space and attract each other when they co-occur. Because of this attraction, concepts that

co-occur frequently tend to cluster. The Euclidian distance between concepts is a measure for their relatedness.
A general criticism on co-occurrence approaches is that although they find relations they cannot say anything about their nature. No distinction can be made between different functional and hierarchical types of relations. This is true for all co-occurrence systems discussed above. Another point is that concepts co-occur, not because they are themselves related, but because they are both related to a third concept, and only to that third concept. Therefore, the central assumption that co-occurring concepts are also related is debatable.
This approach will, nevertheless, be chosen. The reason is that it might be true for a single document that two concepts that co-occur in that document are unrelated. Two concepts that co-occur relatively frequently, however, are unlikely to be unrelated. The greater the number of co-occurrences, the smaller the impact of spurious instances of co-occurrence will be. Moreover, in this thesis we are also interested in indirectly connected concepts. Statistical co-occurrence based systems are used.

2.4 Conclusion

In the preceding section some of the methods found in previous work were discussed briefly. Advantages and disadvantages of both NLP and co-occurrence approaches were treated. In this section, I will establish a list of requirements for an improved system.
The goal of this thesis is helping researchers find relations between concepts of which they were previously unaware. This is most frequently the case when the relation to be discovered is partially described in a research area with which the user is unfamiliar, or where the amount of information is overwhelming. A first requirement is that the system must be able to handle concepts and articles from different research areas.
Systems supporting this kind of search must not demand too much prior knowledge from the users, because these users are trying to find concepts and relations in a field different from their own. This brings us to a second requirement: the system must not call for in-depth knowledge.

In order to find surprising links, the number of articles that the system is able to process must be large. Because of the limited capacity, both in terms of quantity and diversity, NLP is not a viable option for this project.

Co-occurrence techniques can handle more diverse and greater numbers of articles within a reasonable time and therefore seem to be a better choice. The fact that Swanson's findings have indeed sparked research in the medical field increases confidence in this type of methodology.

The discovery system will go on from the ideas developed in the statistical field. The central notion that co-occurrence implies some sort of relatedness will be used. In this thesis I will try to exploit information available in articles on the basis of co-occurrence.

3 Co-occurrence graphs

We are looking for knowledge that is hidden in literature by detecting relationships between concepts in different articles. Two concepts are connected if a relation exists between them A very natural way to model such a problem is by using graphs. This chapter will focus on the graph representation of co-occurrences. However, not all issues are best resolved using co-occurrence graphs. The following chapter is dedicated to those problems.
In this chapter several questions need to be answered in relation to co-occurrence graphs. The first is what kind of co-occurrence we are looking at. The second is how to map those co-occurrences onto a graph. Thirdly we will need to look into how pathways can be found in the co-occurrence graphs. At the end of the chapter problems will remain, due to the limitations of the representation in ordinary graphs.

3.1 Co-occurrence

This section deals with co-occurrence graphs. Some previous work will be examined and the problem will be looked at from different angles. All parts of the graph need to be defined. What is a node? What is an edge? What kind of graph best fits our purposes? As we answer these questions the requirements for the data type *graph* will be become clear.

3.1.1 Co-occurrence in previous work

The idea of representing co-occurrences in graphs is not new. In section 2.3.2 we saw that Stapley and Benoit used a graph with genes as nodes and edges between co-occurring concepts if they have a score of 20 or less for the reciprocal of the Dice coefficient (2.1) (16). Veeling and Van der Weerd used graphs for the depiction of co-occurrences in the Reuters database. (17) In their system, nodes are used for words and edges exist between concepts that either co-occur or both co-occur with a single intermediary concept. Both Stapley and Veeling use undirected unweighted graphs for their purposes.

Important for us is that each node denotes a concept, because our goal is not to find synonyms but relations between different concepts. For that reason we will be working with UMLS concepts with one node per one concept.

We are looking for paths that connect nodes. A path *p* is made up of edges *e* from one node c_n to another: c_i ? e_{ij} ? c_j ? …? c_k ? e_{kl} ? c_l. An edge e_{ij} between two concepts c_i and c_j should represent a *significant* co-occurrence. When searching for a path, we will have a preference for edges that have a more significant co-occurrence over ones that have a less significant score. Therefore, two questions need answering. The first is: what is significant co-occurrence? The second is: on what basis should we select edges to add to our paths?

To define what significant co-occurrence is, we first need to state precisely what counts as co-occurrence and what does not. In the previous chapter, different approaches were discussed each with its own definition. In (4) two words co-occur when they appear in the same title. Stapley and Benoit define co-occurrence as the joint occurrence of two gene names in one document, although he used mostly abstracts of the documents (16).

Many others take a different approach. They count co-occurrence only for terms that together appear in a *window* of a certain size. Veeling and Van der Weerd take the whole document into account, but only count co-occurrence within a window of size 50. This means that they only consider pairs of words no further than 50 words apart. (17). Weeber experimented with window sizes ranging from 1 to 120 (15). Schuemie inverted indices from books to form chains of index words in the order in which they appear in the book. On these chains he examined window sizes of 3 and 4 words (7).

All count unordered pairs of two co-occurring terms. This means that the *co-occurrence pairs* (c_i, c_j) and (c_j, c_i) are not distinguished. The reason that the ordering found in a document is not used, is because statistics of concept distribution do not suffice for distinguishing the semantic type of the relation between c_i and c_j.

There is a difference between ‘Mr White’ –‘shoots’- ‘Mr Blue’ and ‘Mr Blue’ – ‘shoots’- ‘Mr White’ because the nature of the ‘shoots’ relation. When the semantic nature of the relation is unknown, ordering is not useful. ‘Mr Blue’ –‘is related to’- ‘Mr White’ is equivalent to ‘Mr White’ –‘is related to’- ‘Mr Blue’, because the semantic nature of the relation is unknown.

Two considerations determine what is counted as co-occurrence. The first consideration is the level at which one wants to look at the text. Terms that occur in a full-text article may be located in different paragraphs that treat diverse subjects. If one counts only words that are relatively close to each other, only co-occurrence of terms that have a high probability of being related will emerge. The cost is however, that certain interesting co-occurrences will not be found because they are not close enough.

This problem is akin to the classical IR question of finding the balance between *specificity* and *recall*. In IR, a system has to find a set of relevant documents R in a set of documents D. For testing the system, experts mark documents manually as relevant to a certain query. The system finds a set of documents D_f. Specificity is defined as (18):

$$\frac{|D_f \cap R|}{|D_f|} \tag{3.1}$$

and recall as

$$\frac{|D_f \cap R|}{|R|} \tag{3.2}$$.

In IR, changing parameters usually results in a trade-off between these two.

Co-occurrence systems look for relations, not for documents. The co-occurrence system has to find report relations D_f from the set of all relations R. In this way specificity and recall can be defined in the same way as in the previous paragraph. As in IR a trade-off exists in co-occurrence systems. Increasing the window size will increase recall, but lower specificity and decreasing it will result in lower recall and higher specificity. What the right balance is depends on the user’s wishes.

The second consideration is one of size and feasibility. Stapley used a relatively small set of terms, namely the gene names in a set of abstracts on DNA repair. In that case it is possible to count all pairs of terms, because the number of combinations of two terms is manageable. The number of combinations m equals $\binom{n}{2}$, where n is the number of different terms. Veeling uses around 60,000 different terms. The number of possible combinations of co-occurring terms is $\binom{60{,}000}{2} \approx 1.8*10^9$. The number of different terms used increases the potential number of different co-occurrence pairs. Reducing the window size reduces the actual number of co-occurrence pairs found. In order to stay well below the upper limit of combinations, many authors choose to reduce their window size thereby cutting the number of co-occurrences generated.
The following two example strings illustrate the effect of the number of different concepts and the window size:
Let a string s of size n be an ordered list of n concepts c_i:

$s = \{c_i,\ldots,c_n\}$

Our example strings s_1, s_2 each contain 8 concepts:

s_1 = {apple, cow, pear, carrot, strawberry, cabbage, goat, leek}

s_2 = {fruit, animal, fruit, vegetable, fruit, vegetable, animal, vegetable}

Note, string s_2 contains just 3 different concepts
We define a window w_n of size n, centred around c_c, similar to (7) as the set of n words before and n words after c_c. More formally:

$w_n = \{c_{c-n},\ldots,c_{c-2},c_{c-1},c_{c+1},c_{c+2},\ldots,c_{c+n}\}$

Generating w_2 around c_c = carrot yields:

w_2 = {cow, pear, strawberry, cabbage}

Out of w_n, $2n$ pairs are generated using c_c and a member of w_n.
Out of the previous w_2, the following set of pairs P is generated:

P = {(cow, carrot), (pear, carrot), (strawberry, carrot), (cabbage, carrot)}

The relation between the window size and the number of different pairs is seen in table 3.1.

window size	#different pairs s_1	#different pairs s_2
1	7	3
2	13	3
5	25	3
8	28	3

Table 3.1 Relation between window size and number of different pairs for two strings

Table 3.1 shows that augmenting the window size increases the number of different pairs generated, as does the number of different concepts in the string.
Our goal is to find new surprising links between concepts in a large collection of articles. The number of different concepts under consideration must also be large to make serendipity discoveries possible. In this thesis, concepts from the UMLS metathesaurus containing over 700,000 concepts, will be considered.

3.1.2 Defining co-occurrence

Because of the large number of concepts, looking for co-occurrence in full-text articles is too arduous a task. Something shorter than the full text must be used, which must be as informative as possible. The problem is that in shorter texts UMLS concepts are harder to identify. For this reason using only titles or abstracts is not reliable. Another reason is that they are not informative enough, because they only refer to the main subjects of an article. Collexis Fingerprints offer a good list of UMLS terms found in an article (section 2.2.3). A fingerprint $f_k(l)$ is the set of all pairs of a concept c_i and the associated relevance score r_i, for which the relevance score for document k is higher than the cut-off l.

$$f_k(l) = \{(c_i, r_i)_k, ..., (c_n, r_n)_k\} \qquad (3.3)$$
where $\forall j : l < r_j \leq 1$

A concept c_i is included in fingerprint $f_k(l)$ if $f_k(l)$ contains a pair $(c_j,r_j)_k$ where $c_j = c_i$. More formally:

$$c_i \in f_k(l) \Leftrightarrow \exists j : (c_j, r_j) \in f_k(l) \wedge c_j = c_i \qquad (3.4)$$

Occurrence o_k of c_i in $f_k(l)$ is defined using this definition as:

$$o_k(c_i) \Leftrightarrow c_i \in f_k(l) \qquad (3.5)$$

The number of occurrences of c_i in the set of fingerprints L as:

$$O_L(c_i) = \sum_{k \in L} 1_{o_k(i)} \text{ where } 1_x = \begin{cases} 0 & \text{if } x = \mathit{false} \\ 1 & \text{if } x = \mathit{true} \end{cases} \qquad (3.6)$$

Fingerprints will now be ordered on the basis of r_i, rather than on the order of appearance of the concepts in the text. Using a window within the relevance ordered fingerprint is not useful, because there is no reason why two concepts close to each other in the fingerprint would be more related than ones far apart. Instead all combinations of two concepts are considered. Co-occurrence $?_k$ of c_i and c_j in $f_k(l)$ is defined as:

$$k_k(c_i, c_j) \Leftrightarrow c_i \in f_k(l) \wedge c_j \in f_k(l) \qquad (3.7)$$

The co-occurrence of concept c_i and c_j in set L of fingerprints is defined as:

$$k_L(c_i, c_j) \Leftrightarrow \exists k : k_k(c_i, c_j) \wedge f_k(l) \in L \qquad (3.8)$$

Fingerprint $f_k(l)$ is said to *support* the co-occurrence of c_i and c_j if and only if $?_k(c_i,c_j)$.

The number of co-occurrences in the set of fingerprints *L* is defined as:

$$\mathrm{K}_L(c_i, c_j) = \sum_{k \in L} 1_{k_k(c_i, c_j)} \qquad (3.9)$$

Now that co-occurrence has been defined, significant co-occurrence needs to be elaborated upon.

3.1.3 Co-occurrence relevance

The *significance* or *relevance* score of co-occurrence is calculated to estimate the confidence that the co-occurrence reflects a real relation in the literature. Different scores are used in the articles discussed in chapter 2. In (16) the reciprocal of the Dice coefficient (2.1) is used. In (17) only co-occurrences of words no more than 50 words apart are considered. The relevance metric for co-occurrence of words *i* and *j* depends on the distance between occurrences of *i* and *j*, the number of occurrences of *i* and *j*, and the number of co-occurrences.

In this system, relevance ordered fingerprints are used, so distance within a fingerprint is not correlated with relatedness. Therefore, distance cannot be incorporated in our calculation of the significance. Using only the number of occurrences and co-occurrences seems reasonable here.

One commonly used measure is the reciprocal of the Dice coefficient already mentioned in 2.3.2, which was defined as (16):

$$\frac{K_L(c_i, c_j)}{O_L(c_i) + O_L(c_j)} \qquad (3.10)$$

The reciprocal of the Dice coefficient is a reasonable standard, because it gives a measure of co-occurrence dependent on both the frequency of occurrence and of co-occurrence. It can be improved, however, by using a more general measure, that incorporates not only co-occurrence of c_i and c_j but calculates the co-occurrence confidence of c_i and c_j given all other co-occurrences. The discussion of a dependent relevance metric will be continued in the next chapter.

3.1.4 Co-occurrence and hierarchy

Co-occurrence can also be looked at with the knowledge of the hierarchy present in UMLS. This hierarchy represents 'is-a' relationships between concepts. If c_i 'is-a' c_j then c_i is a *child* of c_j. Descendants are children or the generations below children. If c_i is child of c_j or if c_i is a child of a descendant of c_j, then c_i is itself a descendant of c_j. Counting just the co-occurrence of c_i and c_j overlooks the fact that sometimes c_i is not used in a fingerprint, but c_i's children are. When c_j is also present there is co-occurrence between specific instances of c_i and c_j. Because c_i itself is not present this is not counted.

An example is co-occurrence of malaria and anti-malarials. An article that deals with one kind of malaria, e.g. malaria falciparum, and chloroquine will not add to the co-occurrence count of general malaria and chloroquine. The fact that a child of malaria is related to a certain anti-malarial, means that malaria in general is related to that medication. It would be better therefore if the system added the co-occurrences of the concepts not only to the counts of the concepts themselves, but also to those of their parents. This is the idea used in child co-occurrence or descendant co-occurrence measures, described in the remainder of this subsection.

Child co-occurrence $?^c$ is defined as:

$$k_k^c(c_i,c_j) \Leftrightarrow \exists c_p,c_q \in f_k(l) : h_i(c_p) \wedge h_j(c_q) \qquad (3.11)$$

where $h_i(c_j)$ holds $\Leftrightarrow$ c_j is a child of c_i or $c_j = c_i$.

This idea can easily be extended to include not just children, but all descendants of the concepts in question. Descendant co-occurrence is defined as:

$$k_k^d(c_i,c_j) \Leftrightarrow \exists c_p,c_q \in f_k(l) : g_i(c_p) \wedge g_j(c_q) \qquad (3.12)$$

where $g_i(c_j)$ holds $\Leftrightarrow$ c_j is a descendant of c_i or $c_j = c_i$ itself.

Using this for the relevance calculation would improve the reliability of that metric. Counting is handled in a similar way as in simple occurrence:

$$o_k^c(c_i) \Leftrightarrow \exists c_q \in f_k(l) : h_i(c_q) \qquad (3.13)$$

and

$$o_k^d(c_i) \Leftrightarrow \exists c_q \in f_k(l) : g_i(c_q) \tag{3.14}$$

The co-occurrence measure is then defined as:

$$\mathrm{K}_L^c(c_i, c_j) = \sum_{k \in L} 1_{k_k^c(c_i, c_j)} \tag{3.16}$$

and

$$\mathrm{K}_L^d(c_i, c_j) = \sum_{k \in L} 1_{k_k^d(c_i, c_j)} \tag{3.18}$$

The derivation of the equivalent of the reciprocal of the Dice coefficient of c_i and c_j can be calculated as:

$$\frac{\mathrm{K}_L^c}{O_L^c(c_i) + O_L^c(c_j)} \tag{3.20}$$

and similarly

$$\frac{\mathrm{K}_L^d}{O_L^d(c_i) + O_L^d(c_j)} \tag{3.21}$$

These measures possibly improve the metrics. Which to choose still has to be determined by tests with real documents.

3.1.5 Choosing between paths

The relevance metric is not the only measure that requires specification. The next problem is assessing paths in a graph. When multiple paths between non-neighboring concepts exist, a choice has to be made. The preference for a path depends on several factors. The first is the number of nodes that are passed along the way. The more nodes a path consists of, the less reliable it is. A second is the co-occurrence relevance score. Of two paths that consist of an equal number of nodes, the one with higher relevance scores will be preferred. We will have to establish a balance between the two to be able to answer the question whether we prefer a weak two-step path or a highly relevant three-step path, as is illustrated in the following example in Fig. 3.1.

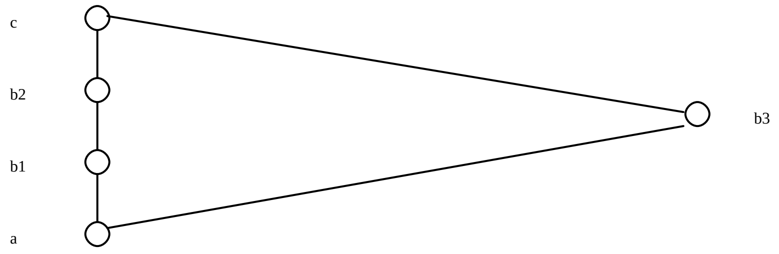


Figure 3.1 Example of a co-occurrence graph

When looking for a path between *a* and *c* in Fig. 3.1, we have the choice between a path a-b_1-b_2-c and a path a-b_3-c. Looking at just the number of nodes, the latter is preferred. Judging on the basis of total edge length the former is better. The appropriate choice for a weighing of both factors has to be found through experiments with users. In chapter 4 more attention will be paid to this problem.

Until now we have discussed what the nodes are made up of, what the edges stand for and what type of weights are assigned. Another question is that of *direction*. We have already seen in section 3.1.1 that co-occurrence itself neither yields information on the characteristics of a relation nor on its direction. The co-occurrence graph is therefore an undirected graph.

Before moving on to the construction of co-occurrence graphs, there is a final refinement we need to add to the graph. Each edge represents co-occurrences between two concepts. Whatever the threshold or relevance metric we choose, if an edge exists co-occurrence exists in the fingerprint set. It is in the interest of the user that references to the articles in which the co-occurrence takes place are stored for future use. A user exploring the graph will want to check on what literature a certain edge is based, so it is important that a reference, e.g. an identification code, is stored.

The graph suggested in this section is an undirected, weighted graph. The nodes stand for concepts and the edges for co-occurrence relations. The edges have attributes that refer to the articles in which the two neighboring concepts co-occur. A hypothetical example of the kind of graph envisioned in this section is found in Fig. 3.2.

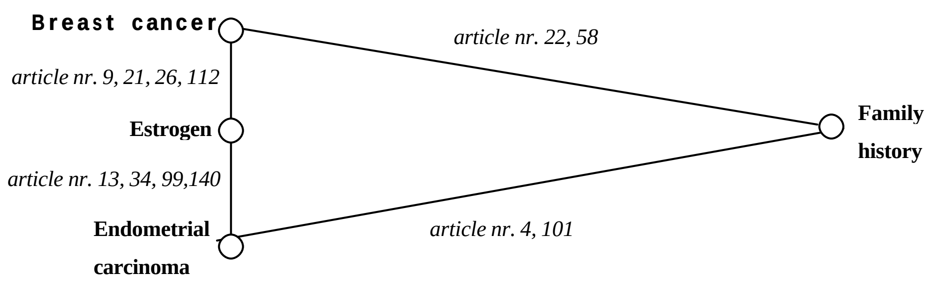


Figure 3.2 example of a co-occurrence graph

In this graph concepts from articles on cancer constitute the nodes and the edge stand for co-occurrence relations. Identifiers of the articles in which the concepts co-occur facilitate retrieval of the literature supporting the relation. A graph like the one in Fig. 3.2 makes the goal of the thesis possible, because in this graph paths between concepts are possible. In the graph nodes and edges can be identified, sometimes also called vertices and arcs. Each edge has a weight, cost or length assigned to it. These terms are treated as equivalent in the thesis.

3.1.6 Graph data structure requirements

The data type *graph* used for implementing the co-occurrence graph must contain functionality to handle all the requirements stated in the previous section. The functionality is listed in table 3.1. In this table CUI stands for Concept Unique Identifier, which is a unique number UMLS assigns to a concepts.

class Graph	
InsertVertex(**int** CUI)	Inserts vertex identified as CUI
InsertEdge(**int** CUI_1, **int** CUI_2)	Inserts an edge between CUI_1 and CUI_2
InsertEdgeAttribute(**int** CUI_1, **int** CUI_2, attribute)	Adds an attribute to the edge between CUI_1 and CUI_2
Adjacent(**int** CUI_1, **int** CUI_2)	Returns whether CUI_1 and CUI_2 are neighbors
Neighbors(**int** CUI)	Returns list of neighbors of CUI

SupportingLiterature(**int** CUI_1, **int** CUI_2)	Returns a list of references which support the edge between CUI_1 and CUI_2

Table 3.1

This graph has been implemented in Microsoft Visual C++ 6.0 using an edge list representation adapted from (19). This implementation allows graphs to become large without excessively taxing memory as in an adjacency matrix or list representation. The complete code is listed in appendix B. The next section will treat the construction of such a graph on the basis of a set of articles.

3.2 Mapping co-occurrences onto a graph

In the previous section we have introduced the idea of co-occurrence graphs based on co-occurrence of UMLS concepts in conceptual fingerprints. In this section the mapping from fingerprint to graph will be treated in greater detail. A test set used in the following chapters will be introduced and some experiments conducted.

Before moving on, some recapitulation first: section 2.2.3 already gave an introduction to fingerprints. These fingerprints are used for the construction of the co-occurrence graph. Two concepts co-occur if they are jointly present in at least one fingerprint. We will only count the most relevant concepts in a fingerprint using the relevance metric *r*. When the *r* of both c_i and c_j is greater than the threshold *l*, a co-occurrence is counted. The co-occurrence of c_i and c_j gives rise to an edge in the co-occurrence graph G.

Figure 3.3 shows an example of a set of fingerprints. In this example the *r* = 1.0 for C0026237 and *r* = 0.9 for C0177337.

$$\begin{bmatrix} C0026237 & 1.0 \\ C0177337 & 0.9 \\ C0012634 & 0.3 \end{bmatrix}_1 , \begin{bmatrix} C0442996 & 1.0 \\ C0521451 & 0.8 \\ C0032659 & 0.6 \end{bmatrix}_2 , \begin{bmatrix} C0012634 & 1.0 \\ C0177337 & 0.9 \\ C0032659 & 0.6 \end{bmatrix}_3$$

Figure 3.3 examples of fingerprints

An $f_k(l)$ of size n generates $\binom{n}{2}$ co-occurrence pairs. Each pair is an unordered set of two concepts and is counted for calculations of the relevance metric. Because the tail of the fingerprint often contains erroneous concepts, it is best to cut off all concepts with relevance score lower than a certain cut-off value l. Experiments are necessary to establish a good value for l. Setting l = 0.7 in Fig. 3.3 would cut off the bottom concept of all three fingerprints. l = 0.7 on these fingerprints results in the co-occurrence graph in Fig. 3.4.

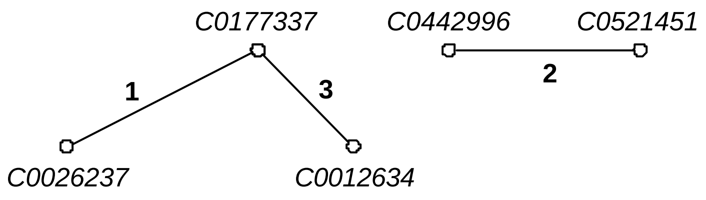


Figure 3.4 Co-occurrence graph of Fig. 3.3

Experiments throughout this thesis have been done on one particular set of fingerprints. These fingerprints stem from three years of the journal Nature Genetics (20). The articles all pertain to the domain of genetics. The set contains 708 fingerprints.

The number of distinct concepts generated from this set of fingerprints depends on the cut-off l. The higher l, the lower the number of concepts in the graph. For Nature Genetics the relation is as follows:

l	#nodes	#edges	Branch. factor
1	0	0	0
0.9	924	2235	4.84
0.8	1167	4071	6.98
0.7	1315	6173	9.39
0.6	1424	8394	11.79
0.5	1594	12313	15.45
0.4	1827	20534	22.48
0.3	2223	38917	35.01
0.2	3242	127956	78.94

When setting the cut-off at $l = 0.4$ the number of unique concepts extracted from the article set drops to 1827. The number of distinct co-occurrence relations at $l = 0.4$ is 20,534. In graph terms: the cardinality of the graph is 1827 and the number of edges 20,534. The average number of edges incident with each node, or the branching factor, is therefore 22.48.

3.3 Searching for paths in a co-occurrence graph

The graph as we defined it in table 3.1 serves two purposes: finding neighbors of concepts for exploratory purposes and finding paths between concepts between which a hypothetical relation exists. Finding the neighbors of an arbitrary c_i is a straightforward task, using the edge list. The graph implementation simply returns a list of all the concepts found in the edge list of c_i. Finding a path between two arbitrary nodes in a graph is a clearly defined problem in discrete mathematics. In this paragraph we will discuss the algorithms developed for finding shortest paths in a weighted graph.

The best-known algorithm for finding shortest paths is certainly Dijkstra's algorithm. Several algorithms have been developed since Dijkstra first proposed his well-known searching algorithm. Most others improve on the original algorithm, but the principal strategies of search are usually the same. Dijkstra's algorithm will therefore be discussed first and most extensively. Then an algorithm of a very different type will be presented. This is the A* algorithm which improves on Dijkstra's use of heuristic information.

3.3.1 Dijkstra's shortest path algorithm

The problem for the Dijkstra algorithm is designed to deal with the single-source shortest path problem. This problem is formulated as (21):

Single-source shortest path problem

Given as input a weighted graph, G = (V,E), and a distinguished vertex, s, find the shortest weighted path from s to every other vertex in G.

This is not entirely the same problem as we are dealing with, because we are not looking for paths to every vertex from a source, but to one specific target vertex t. Obviously, if one finds the shortest paths between a source and every vertex, the shortest path between the source and the target t is also found. The problem is rather that the algorithm does too much work. This problem will be discussed in later in this section.

The Dijkstra algorithm will be shown using pseudo code and in text. In pseudo code it looks like this:

```
For all vertices $v$ set status 'unknown' and $d_v = 8$. Set $d_s = 0$.
While(remain unknown vertices)
        select $v$ with smallest $d_v$ from all vertices marked 'unknown'.
        mark $v$ as 'known'.
        For all neighbors $w$ of $v$ with edge weight $c_{v,w}$ do
            if ($d_v + c_{v,w} < d_w$) then
                $d_w = d_v + c_{v,w}$.
                $p_w = v$.
```

The pseudo code is explained in the following paragraphs. The Dijkstra algorithm stores several kinds of information during the search. It marks every vertex as either 'known' or 'unknown'. A tentative distance d_v is kept for each vertex v. Distance is the sum of the weights of the edges that together form the graph. This distance becomes the definite shortest path length from s to v when it uses only known vertices as intermediates. The vertex that last caused d_v to change is stored in p_v. The p_v can later be used to construct the path from s to t.

The algorithm proceeds in stages. At each stage it selects vertex v, which has the smallest d_v among all unknown vertices, and declares v 'known'. Dijkstra's algorithm therefore selects the best local solution at every stage, making it a *greedy* algorithm. In the remainder of this stage, the values of the other d_w are updated.

This updating takes place by setting $d_w = d_v + c_{v,w}$ if $d_v + c_{v,w} < d_w$. Or, in other words, we decide whether it is a good idea to use v in the path to w. The resultant d_w are the cheapest costs of paths from s to w using all known vertices, including v. If d_w is updated in this stage, p_w is set to v. Setting p_w to v means that on the shortest path from s to w found so far v is the predecessor of w.

In the beginning all v except s are assigned d_v = 8, and for s $d_s = 0$. The first v selected is s and from here the next v is chosen. The algorithm continues until v has no more unknown neighbors. The algorithm terminates when there are no 'unknown' vertices left.

The result is a graph with known and correct shortest path length d_v for each v. Paths from s to v can be constructed by retracing the algorithm through the p_v. This is done by starting at t and adding the vertex $v = p_t$ before t in the path. p_v The vertex is then placed before v and so on until the s has been added to the path and it is complete.

The assumption behind the algorithm is that the graph does not contain *negative cost cycles*. These cycles have a negative total length. Because the co-occurrence graph never includes negative cost edges, such cycles do not occur and the assumption is satisfied.

An example of Dijkstra's algorithm is given in Fig. 3.5. In this example the known vertices are marked with an asterisk. The weights of the edges are written in bold along the edges. The d_v of each vertex at each phase is found on the right hand side of each node. The source vertex is v_0.

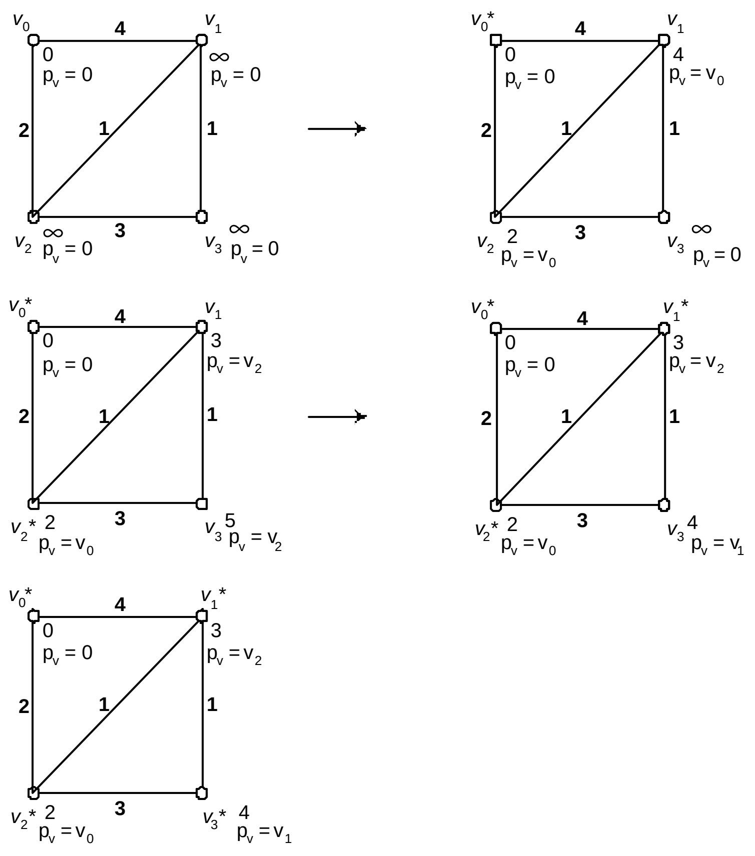


Figure 3.5 the Dijkstra algorithm

In the top left corner the initial state is depicted. All nodes are unlabeled and all d_v are set to 8, expect d_{v0}. In the first stage v_0 is selected and set to known. All other nodes updated. v_2 is still unknown and has lowest d_v and is therefore selected for the next stage. After updating, v_1 is selected and finally v_3.
The result is a situation where all vertices are known and each d_v denotes the length of the shortest path from v_o. With the p_v the paths from the each v to s can be retraced. For example the path from the v_0 to v_3 can be constructed by successively adding the p_v before v_3. This result is $v_0 - v_2 - v_1 - v_3$.

The time bound on Dijkstra's algorithm depends on how vertices are manipulated. With a straightforward array implementation the time bound is O(|V|^2), where |V| is the cardinality of V. Using pair heaps the bound is O(|E| log |V|) where |E| is the number of edges in the graph. With the vertices stored in Fibonacci heaps the time bound is O(|E| + |V| log |V|) . Depending on the density of edges one or another implementation is more efficient (21).

Other implementations have been suggested with lower time bounds (22). All variants of Dijkstra's algorithms have the problem that all vertices are assessed. Because the aim of the system is not to find the distance to every other vertex, but only from s to t, too many vertices are assessed. Terminating the algorithm as soon as the target node t is found reduces the amount of work, but is not an efficient way to search for one shortest path.

The algorithms of the single-source shortest path variety would be time-consuming for searching paths from v to t in the co-occurrence graphs of the size this research aims at. The reason is that at every step the best local solution is selected. Setting optimal local steps in the graph does not mean one has gotten closer to t.

The speed of the search would improve greatly if at every phase information about the remaining distance from w to t would be available. The w from the shortest path to t would be selected at every step, until t is reached and known. The information is not available, because running the algorithm would not be necessary then.

Although the exact information about the remaining distances is not available in any relevant graph, heuristic information is sometimes present. An example of such heuristic information is the 'as the crow flies' distance on maps, which is an indication of the distance between two points for a car. The A* algorithm uses this type of heuristic information.

The A* algorithm can look like this in pseudo code:

```
General rule $f_v = d_v + h_v$ holds throughout algorithm, where $h_v$ is the heuristic estimate of
    the distance from $v$ to $t$.
For all $v$ set status 'unknown' and $d_v = 8$. Set $d_s = 0$ and mark $s$ as 'temporary'.
Select temporary marked $v$ with lowest $f_v$. If $t$ is among the vertices with minimal $d_v$, select $t$.
While($v$ ? $t$):
    mark $v$ as 'known'.
    for all neighbors $w$ of $v$
            if ($d_v + c_{v,w} < d_w$) then
                    $d_w = d_v + c_{v,w}$.
                    mark $w$ 'temporary'.
                    $p_w = v$.
```

The algorithm keeps more data than the Dijkstra class of algorithms. It marks vertices as 'unknown', 'temporary' or 'known'. For every *v* the estimated length of the path f_v from *s* to *t* via *v* is stored. f_v is the sum of the length of the shortest path from s to v, d_v, and h_v the heuristic estimate of the length of the remaining path to *t* from *v*.

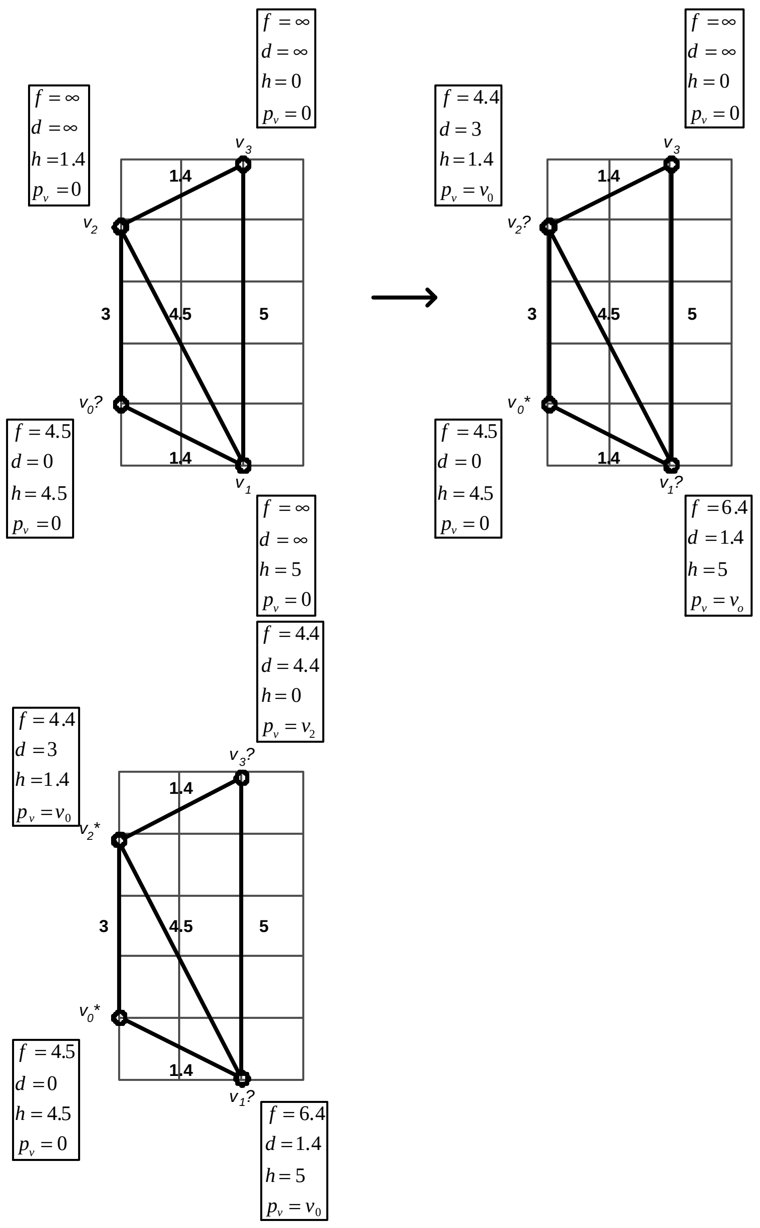
$$f_v = d_v + h_v \qquad (3.22)$$

At every stage the *v* with the smallest f_v from all *v* with a temporary label is selected. If more than one vertex exists with minimal f_v and *t* is one of them we select *t*. *v* is set to known. The status of all neighbors *w* of *v* is then updated.
The updating proceeds as follows. If *w* is unknown it is assigned a temporary label and $d_w = d_v + c_{v,w}$. The estimated path length f_w changes to $f_w = d_w + h_w$. If *w* has a temporary label then $d_w = d_v + c_{v,w}$ if $d_w > d_v + c_{v,w}$. If *w* is known and $d_w > d_v + c_{v,w}$ then $d_w = d_v + c_{v,w}$ and the label is changed to temporary. In all cases where d_w changes p_w is set to *v*.
The initialization step is to assign $f_s = g_s + h_s = h_s$ and $f_w = \infty$ to all other vertices. Source vertex *s* is assigned a temporary label and all other vertices the unknown label.

The algorithm terminates with an error when at the beginning of a stage no temporary labeled vertices are left, and with success when the *v* selected equals *t* (23).

How the A* algorithm works is illustrated in Fig. 3.6. In this figure the search is for the shortest path between v_0 and v_3. Each vertex has an absolute position in the grid. The heuristic information is the Euclidian distance in the grid.

Known vertices are marked with an asterisk. A temporary label is indicated with a question mark. For every *v* the f_v, d_v and h_v are found in the boxes next to the vertex.



$f=\infty$ $d=\infty$ $h=0$ $p_v=0$
$f=\infty$ $d=\infty$ $h=1.4$ $p_v=0$
v_3
v_2
1.4
3
4.5
5
$v_0?$
$f=4.5$ $d=0$ $h=4.5$ $p_v=0$
1.4
v_1
$f=\infty$ $d=\infty$ $h=5$ $p_v=0$
$f=\infty$ $d=\infty$ $h=0$ $p_v=0$
$f=4.4$ $d=3$ $h=1.4$ $p_v=v_0$
v_3
$v_2?$
1.4
3
4.5
5
v_0*
$f=4.5$ $d=0$ $h=4.5$ $p_v=0$
1.4
$v_1?$
$f=6.4$ $d=1.4$ $h=5$ $p_v=v_o$
$f=4.4$ $d=4.4$ $h=0$ $p_v=v_2$
$f=4.4$ $d=3$ $h=1.4$ $p_v=v_0$
$v_3?$
v_2*
1.4
3
4.5
5
v_0*
$f=4.5$ $d=0$ $h=4.5$ $p_v=0$
1.4
$v_1?$
$f=6.4$ $d=1.4$ $h=5$ $p_v=v_0$

Figure 3.6 the A* algorithm

In Fig. 3.6 the first grid is the initial state of the algorithm. The second shows the end of the first stage with all updating done. The third grid shows the end of the second stage, after which the program select $v = v_3$ and terminates.
It is interesting to note that the A* algorithm is faster than the Dijkstra algorithm would have been. Where A* selects v_0, v_2, v_3 and terminates, Dijkstra's algorithm chooses v_0, v_1, v_2, v_3 for the stages. The reason is that v_1 is a local optimum and a greedy algorithm like Dijkstra's always selects local optima.

If the heuristic estimate h is always smaller than or equal to the actual path length between two points, the optimal solution will be found. If h is sometimes larger than the actual length a guarantee that the optimal global solution is found cannot be given. Just as for Dijkstra's algorithm the computation time depends on the data structures used for the implementation of the lists of vertices. If Fibonacci heaps are used to store the vertex lists, the heuristic algorithm will inspect fewer vertices than Dijkstra algorithms do (23). It is therefore faster for finding paths between two points in a graph. The problem is that with the co-occurrence graph no such heuristic information is available to the algorithm. This matter will be the subject of the next chapter, where we will try to find such heuristic information from the co-occurrence data.

3.4 Conclusion

In section 3.1 of this chapter three kinds of co-occurrence were defined for use in the remainder of this thesis. The first was simple co-occurrence, while the others used thesaurus information. Which one to choose is a difficult question, especially because all failed in one respect: they do not take into account other co-occurrences than the one in question. The question concerning which of these should be used is deferred to chapter 4. The basic requirements and design of the co-occurrence graph were also introduced. In section 3.2 the issue of mapping the fingerprints to a co-occurrence graph was discussed and a simple test with the Nature Genetics set was set up. In section 3.3 the disadvantages for our purposes of conventional searching where shown. With a large number of nodes and edges the time required for finding a path between two nodes grows rapidly. A* was suggested as an improvement, but it cannot be used without extra information in the graph.

Therefore, how to measure the lengths of paths and how to search in the co-occurrence graph still needs to be decided. The next chapter will introduce a way to do both.

4 Associative Conceptual Space

In chapter 3 we looked closely at co-occurrence graphs and finding paths in these graphs. The problems we discovered were those of deciding on preference for paths, of deciding on the right distance measure and the problem of navigating in a large graph. To deal with these issues an Associative Conceptual Space (ACS) is introduced in this chapter.

The central ideas behind the ACS as introduced by Schuemie are discussed in 4.1. The use of the ACS for the research goal is the subject of 4.2. The construction of an ACS is subject of inquiry in 4.3. In 4.4 experiments are conducted to test the ACS implementation. Chapter 4 concludes with 4.5 in which we describe the uses of ACS for our search of paths.

4.1 Principles of Associative Conceptual Space

In chapter 3 we used a graph to depict the structural relations between concepts. Edges, representing relationships, exist between nodes representing concepts, when concepts co-occur in at least *n* documents in the set. Graphs are one way of viewing relations, another was suggested by Schuemie: Associative Conceptual Space (7). This section first discusses the goal of the ACS (4.1.1), then how Schuemie constructs the Associative Conceptual Space (4.1.2).

4.1.1 Using ACS for Information Retrieval

Schuemie's purpose was to develop a system for information retrieval that would perform better than keyword-based search algorithms by using semantics. The knowledge used for information retrieval is gathered from the indexes of natural language books. This subsection gives a summary of how the indexes are used to construct a representation of knowledge in a domain.

The end result of the ACS is a two-dimensional colored surface where each document has a place and related groups of documents appear in clusters. Figure 4.1 is an example of such a map of (Dutch) categories.

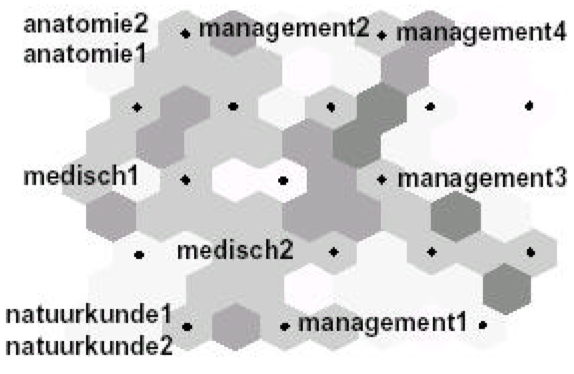


Figure 4.1 ACR Map of search terms (7)

In Fig. 4.1 related categories appear more or less together: management 1 to 4 on the right hand side and medicine, anatomy and physics on the left. In the following we will describe how indexes are used to create maps like the one in Fig. 4.1.

These maps form the interface for an Information Retrieval system that assists the user in choosing fields of interest, not just single keywords. How this map is constructed from indexes is discussed in the next subsection. Before moving on, it is important to note that Schuemie does not use predefined concepts in his knowledge representation as we do, but index terms. The use of index terms solves the problem of synonyms within one book, but not across books. The index terms chosen in one book may be synonymous to index terms in another book, but they will not be treated as one by the algorithm, which has no way to compare terms.

4.1.2 Schuemie’s construction of ACS

The input for the ACS algorithm is a set of indexes. Indexes are usually alphabetically ordered and this means that any semantic relation that was present in the book has been lost. Indexes are still useful because they contain only informative terms and that is the reason they are used in this algorithm. Not all order information has been lost irrevocably because after the term a list of pages on which the term can be found is given. To recover some of the original semantics a preprocessor inverts the indexes using the page numbers. This inverting is the process where a list of index terms is constructed by placing the terms in the order of the pages they were originally found. If a term is found on multiple pages it appears on several places in the *inverted index*. This inverted index reflects the book because terms that appear in each other's vicinity in the book are also located close to each other in the inverted index. The inverted indexes of all the books are then concatenated to form one list of terms.

The list of index terms from different books is used to create an *n*-dimensional Euclidian space. In this space each term is assigned an *n*-dimensional reference vector, which is its location in space. In the ACS Euclidian distance between the reference vectors of two terms is the measure for their relatedness. It is important to note that the dimensions of the space do not represent anything as such.

Assigning reference vectors to the terms is a non-trivial problem, because all distances between vectors measure strength of the corresponding association. To deal with this problem the vectors are *trained* using artificial intelligence techniques. Two vectors are brought closer to each other if the terms appear in each other's neighborhood in the inverted index list. The exact way pairs of terms are selected will be explained in greater detail in the last part of this subsection.

The greater the number of dimensions the easier it is to train the ACS, because the number of degrees of freedom is greater. These degrees of freedom come at a computational price, in the sense that each epoch takes longer. Convergence is reached at an earlier epoch, so the most efficient trade-off needs to be found.

The data is fed to the algorithm several times and the learning rules are intended to change the position according to the data. Every time the data is entered the algorithm enters a new stage called an *epoch*.

After starting at random positions the distance between reference vectors is reduced using the normalized Hebbian learning rule (24). The *association rule* for reference vectors x_i and x_j is:

$$x_i(t+1) = x_i(t) + h(t)\frac{x_j(t) - x_i(t)}{\left\| x_j(t) - x_i(t) \right\|} \tag{4.1}$$

$$x_j(t+1) = x_j(t) - h(t)\frac{x_j(t) - x_i(t)}{\left\| x_j(t) - x_i(t) \right\|} \qquad (4.1)$$

where $h(t)$ is the learning rate.

$h(t)$ is the fraction by which the vector distance is changed. This fraction can be set at a constant. Another option is to reduce $h(t)$ with the number of epochs, until it has reached a lower bound on the learning rate. This can be expressed as:

$$h(t) = \frac{2}{\min(t,u)} \qquad (4.3)$$

where $\frac{2}{u}$ is the pre-specified lower bound on *?(t).*

The bound u keeps *?(t)* from becoming very small. Small *?(t)* are a problem because then the learning does not move the terms enough to offset the forgetting, introduced later this section.
Schuemie uses both a constant and variable *?(t)* in his experiments. The reasoning behind the variable *?(t)* is that in the beginning large movements in the space are necessary, because in the beginning the distribution of concepts is far off from the desired distribution, where distance reflects relatedness. After the initial positioning all that remains is fine-tuning.

This learning mechanism is further refined to deal with more and less informative words:

$$x_i(t+1) = x_i(t) + h(t)\frac{x_j(t) - x_i(t)}{\left\| x_j(t) - x_i(t) \right\|} a(i,j) \qquad (4.4)$$

$$x_j(t+1) = x_j(t) - h(t)\frac{x_j(t) - x_i(t)}{\left\| x_j(t) - x_i(t) \right\|} a(i,j) \qquad (4.5)$$

where $a(i,j) = \dfrac{2\,freq_*}{freq_i + freq_j}$

and where *freq*$_i$ is the frequency of term *i* in the string of all terms and *freq*$_*$ is the average frequency of a single term.

By this mechanism, less frequent words have a greater impact than more frequent ones. This learning mechanism is illustrated in Fig. 4.2.

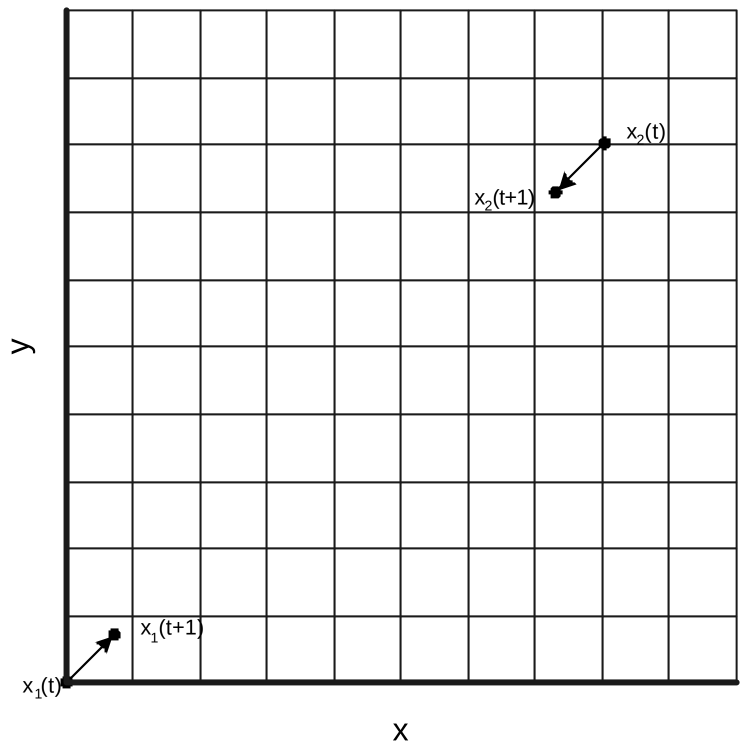


Figure 4.2 Movement of two vectors following the association rule

In Fig. 4.2 two points are moved using the learning rule.

This learning will cause a space to implode because the learning rule moves terms only towards each other. To balance this Schuemie uses a *forgetting rule*. After each learning phase an active forgetting rule brings relief. The rule is applied on all distinct pairs of terms *i* and *j*.

$$x_i(t+1) = x_i(t) - l\left(\left\|x_j(t) - x_i(t)\right\|\right)\frac{x_j(t) - x_i(t)}{\left\|x_j(t) - x_i(t)\right\|} \qquad (4.6)$$

and

$$x_j(t+1) = x_i(t) + l\left(\left\|x_j(t) - x_i(t)\right\|\right)\frac{x_j(t) - x_i(t)}{\left\|x_j(t) - x_i(t)\right\|} \tag{4.7}$$

where $l(x)$ is the repulsion function. $l(x)$ is defined as:

$$l(x) = \begin{cases} 1 & for\ x < 1 \\ 1/x & for\ x \geq 1 \end{cases}$$

Now that we have introduced the general idea we take a closer look into which terms are activated together. The basis for the co-activation is the inverted index.

For selection of terms for co-activation a neighborhood of size 2 or 3 is defined as in Section 3.2.2. From these neighborhoods pairs of terms are generated and are fed to the learning rule.

After all pairs have been fed to the ACS, the forgetting phase begins and every combination of two terms is repulsed using the forgetting function. This process of learning and forgetting is repeated with a decreasing learning rate $h(t)$. After training related terms appear close to each other in the space.

Schuemie has no real criteria for when to stop training. This is a problem that needs to be dealt with. He stops after around 100 epochs, because no big changes seem to occur in the word clusters.

The ACS is of most interest to us, so how Schuemie proceeds from here is treated only briefly: the reference vectors of each term are the input for Kohonen's Self-Organizing Maps dimension reduction algorithm that returns *n* clusters of terms (25). Each document *d* is then described as an *n*-dimensional document-vector. A value greater than zero is assigned to a component of the vector for every cluster that contains a word from the index of *d*. These document vectors are then the input of a document-clustering algorithm. The result of this final clustering is a grid of document clusters from which the user can select the most relevant one for his search.

In the next section, the use of ACS for this research will be developed.

4.2 Using ACS for concept representation

In the previous section we have seen that terms can be placed in an *n*-dimensional space. Each term is assigned one coordinate. On the basis of these coordinates the

Euclidian distance between two points can be calculated. The distance between two terms is an indication of relatedness, where relatedness depends on the number of co-occurrences.

Adapting Schuemie's ACS algorithm to the goals of this thesis is not a difficult exercise. Where he uses inverted indexes, fingerprints can be substituted. Where the ACS deals with terms, concepts can be used. All relevant c_i in a set of articles, that is, with at least one $r_{i,k} = l$, can be assigned a coordinate. An adapted version of ACS can then be used to train the space, until the distances in the space represent the relatedness of concepts. This adapted ACS is called Conceptual Associative Spatial Graph (CASG). How the fingerprints are fed to this algorithm and other adaptations will be discussed in the next section (4.3).

In this section the use of the ACS for the problems discovered in earlier chapters will be discussed. Two problems remained from chapter 3. The first one was a good way to assign length or weights to paths and the second was searching for shortest paths between two arbitrary concepts in a graph. The ACS makes possible an interesting tactic to deal both problems. The next two subsections (4.2.1 and 4.2.2) discuss how the ACS fits that purpose.

4.2.1 Edge length and the ACS

In Section 3.1.5 we saw that the problem of assigning weights to edges is a preference problem. When a choice exists between several edges we would like to select the strongest. Strongest means most related, or in other words, the edge that has most support in the article set. To be able to order the edges for preference, a weight or cost is assigned to each edge. The greater the relatedness of two concepts, the smaller the weight of the connecting edge.

In Section 3.1.5 several options for assigning weights were considered, all based on the reciprocal of the Dice coefficient. The problem with these measures is that they give us independent weights. The weight of an edge between c_i and c_j is determined solely by their co-occurrence, without considering the co-occurrence of other concepts in the article set.

A measure that is not based on independent co-occurrences ensures that the weight $w_{i,j}$ of an edge between c_i and c_j depends on the co-occurrences of the sets of neighbors G_{ci} and G_{cj} respectively. The idea behind this is that if concepts related to c_i and c_j co-occur, c_i and c_j are drawn closer to each other. The usefulness of this idea will be discussed first, after which the notion will be developed formally. The last part of this section is dedicated to how the ACR can be used for the distance measure.

The question why a dependent weight assignment scheme is a good idea, needs a more precise answer. The reason is that the frequency of co-occurrence of c_i and c_j is not the only evidence the concepts are related. Co-occurrence of concepts related to c_i and c_j or concepts related to c_j, and vice versa, strengthens the notion that c_i and c_j are related.

Dependent weight assignment is best illustrated with a hypothetical example. A co-occurrence graph is based on an article set that contains articles about two diseases *a* and *b*. Let us say that *a* and *b* co-occur in several articles. On the basis of this co-occurrence some sort of relation is suspected. This suspicion is strengthened if they both co-occur with a certain medicine *d*. The strength of the relation between *a* and *b* is thus greater because of the pairs of co-occurrences *a* with *d*, and *b* with *d*. If *a* and *b* both co-occur with another medicine this further increases their relatedness.

If the co-occurrence of *a* and *c* and *b* and *c* is very frequent then this co-occurrence has a greater impact than when these co-occurrences are infrequent.

For the formalization some definitions from the field of discrete mathematics are needed. In discrete mathematics the notion of *coherent subgraphs* exists (26). A coherent subgraph ? is a subgraph in graph *G* that is fully connected, i.e., a path exists between every pair of nodes in ? using only edges in ?. Two coherent subgraphs are node disjoint if no node *x* exists $x \in g_1$ and $x \in g_2$.

In a co-occurrence graph coherent subgraphs can be identified as well. The notion of coherent subgraphs will be used to formulate some statements about edges *e* and *u* in a co-occurrence graph.

Let *e* and *u* be edges in a co-occurrence graph G^c, and let the frequency of the concepts they connect and the support for the edges be the same for *e* and *t*, then the following statements are properties of a useful dependent weight assignment scheme.

- Let *e* connect node $x \in g_1$ and $y \in g_2$, and let ?$_1$ and ?$_2$ be node disjoint coherent subgraphs in G^c. Let *t* connect node $z \in g_3$ and $u \in g_4$, and let ?$_3$ and ?$_4$ be node disjoint coherent subgraphs in G^c. Let *n* be the number of edges

$g_1,..,g_n$ that connect nodes in ?1 and nodes in ?2 and let m be the number of edges $h_1,\dots,h_m$ that connect nodes in ?3 and nodes in ?4. If $n < m$ then therefore $w_e > w_t$.

- Let the total weight of the edges between ?1 and ?2 be $W_{g_1,g_2} = \sum_{i=1}^{n} w_{g_i}$ and the total weight of the edges between ?3 and ?4 be $W_{g_3,g_4} = \sum_{i=1}^{m} w_{h_i}$. If $n = m$ and $W_{g_1,g_2} < W_{g_3,g_4}$ then $w_e > w_h$.

Now the usefulness of a dependent weighting scheme and its formalization have been demonstrated, the ACS must be inspected. In the beginning of this section it was suggested that the ACS could be used for a dependent weighing scheme.

ACS can be used for assigning weights by setting edge weights to the Euclidian distance between two concepts in the trained ACS. It is necessary to check whether that scheme satisfies the two requirements for a dependent scheme.

The first test is whether co-occurrence of concepts to c_i and c_j decreases the distance between c_i and c_j. If we take a concept c_k that co-occurs with c_i and c_j, the distance between c_i and c_k and the distance between c_j and c_k is decreased using the learning rule. As a consequence c_i and c_j both move towards a certain point in space (the location of c_k) and their distance is decreased. Because $w_{ci,cj}$ is determined by the distance it is decreased as well. This can be extended to any number of intermediary nodes from which a path exists to c_i and c_j. The first requirement is therefore satisfied.

The second requirement is that $w_{ci,cj}$ needs to decrease more when the other connecting weights are smaller. In the ACS weights are decreased in the learning phase for every article in which two concepts co-occur. Every time c_i and c_k or c_j and c_k co-occur, c_i and c_j move closer to each other. The ACS distance weighting also satisfies the second requirement.

Using the distances in the ACS for the edge weights is a good measure for relatedness between concepts. It is more satisfactory than the independent weights. It might be a good idea to add a penalty per node in the path, but that will not be tested in this thesis.

4.2.2 ACS and searching in graphs

In the ACS all concepts are assigned a reference vector and have a location in the space. Between every two concepts the Euclidian distance can be calculated using these reference vectors. In section 4.2.1 the distance between two connected c_i and c_j was used as a weight for the edge between c_i and c_j. When weights are assigned in this way the Euclidian distance between c_i and c_j is a lower bound on the length of the shortest path between them.

Because the Euclidian distance is a lower bound that satisfies the assumption of the A* algorithm, A* will always find the optimal solution in the co-occurrence graph. A* is faster than the Dijkstra class algorithms.

In the previous two subsections the usefulness of the ACS as an addition to co-occurrence graph has been demonstrated.

4.3 Construction of a CASG

Schuemie's method for constructing an ACS was the subject of Section 4.1. Before moving on to the implementation it is necessary to discuss how his ACS algorithm can be adapted and refined. The adapted ACS, called CASG is developed in this section.

As said in Section 4.2, adapting the ACS to the research needs of this thesis is not very hard. Schuemie uses terms from indexes. For the CASG UMLS concepts are used. This does not make a difference for the construction because our input is not made up of terms, but of numbered concepts. For the user there is a difference as concepts are less ambiguous than terms.

The input to CASG will consist of a set of fingerprints and not of inverted indexes. The major difference is in how the order of the concepts or terms is treated.

In Schuemie's algorithm the inverted indexes are concatenated to form one long list ofconsecutive terms. A window is moved over this list and co-occurrences of the central term and all other terms in the window are counted.

In the CASG, the number of concepts in the fingerprint determines the window size. It moves from the first to the last relevant concept in the fingerprint and counts the co-occurrence of each concept c it passes with all concepts after c.

It is not a good idea to add all fingerprints, or indexes, to one list of consecutive terms. Adding them only causes the generation of spurious co-occurrences at the ends and beginnings of fingerprints. Concepts c_i would at the end of one fingerprint co-occur with the first concepts c_j of the next fingerprint.

The input for the CASG is therefore different from Schuemie's ACS. In this approach all concepts within one fingerprint co-occur and no co-occurrence between concepts in one fingerprint and another is counted. After the input the learning and the forgetting rule need inspection.

4.3.2 The learning rule

In Schuemie's thesis the central term in a window co-occurs with all other terms in the window. This means that one by one all pairs of the central concept and another concept are activated and their distance decreased. After all pairs have been activated the center of the window moves to the next term and for that term all pairs are activated. The purpose of these activations is that all terms within a certain distance are moved towards each other.

In the CASG not all terms within a certain distance, but in the same fingerprint are moved towards each other. This can be done by activating every pair of terms, but a more intuitive and computationally less intensive way is to move all concepts towards one point. This way the distance between all concepts is reduced. The best choice for such a point is the imaginary focus of the article or the *central point*. The central point is defined as the average of all vectors of a fingerprint.Each concept c_i is defined as:

$$c_i = (c_{i,1} \quad c_{i,2} \quad \cdots \quad c_{i,n})^T$$

More formally, for $f_k(l)$ containing m relevant concepts, and an n-dimensional ACS in which all relevant concepts have a reference vector, the central point p_k of $f_k(l)$ is defined as:

$$p_k = \begin{bmatrix} p_{k,1} \\ \vdots \\ p_{k,n} \end{bmatrix} = \begin{bmatrix} \frac{\sum_{h=0}^{m} c_{h,1}}{m} \\ \vdots \\ \frac{\sum_{h=0}^{m} c_{h,n}}{m} \end{bmatrix} \tag{4.8}$$

where $c_{h,n}$ is the nth coordinate of concept h.

The calculation of p_k for three concepts in a 2-dimensional space is illustrated in Fig. 4.3.

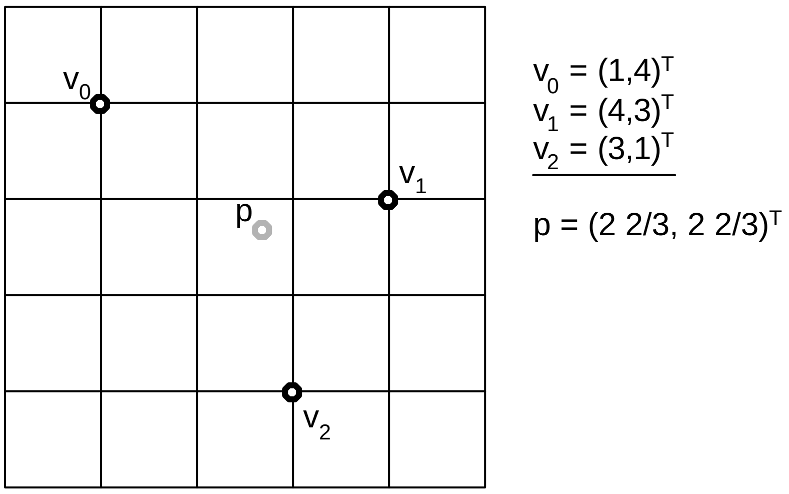


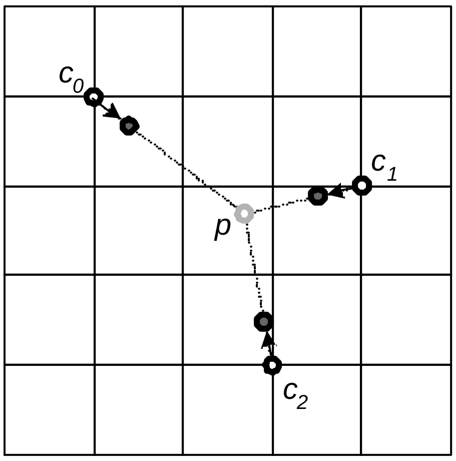
Figure 4.3 Calculation of a central point.

The learning rule is adapted so that each concept is attracted to the central point. This will happen by using the same rule as Schuemie uses:

$$x_i(t+1) = x_i(t) + h(t)\frac{p_k(t) - x_i(t)}{\|p_k(t) - x_i(t)\|} \tag{4.9}$$

This learning rule causes the distance from a concept to p_k to be reduced. All concepts of a fingerprint $f_k(l)$ are moved in the direction of p_k. All points are moved an equal length. How the moving works is illustrated in the following paragraph.

Figure 4.4 shows an example of this rule for an $f_k(l)$ containing three concepts in a 2 dimensional space.



$c_0 = (1,4)^T$
$c_1 = (4,3)^T$
$c_2 = (3,1)^T$

$p_k = (2\ 2/3,\ 2\ 2/3)^T$

Let $h(t) = 0.5$ then

$c_0 = (1.390,\ 3.688)^T$
$c_1 = (3.515,\ 2.879)^T$
$c_2 = (2.902,\ 1.490)^T$

Figure 4.4 New positions using the learning rule

The calculation for c_0 is as follows:

$$c_i(t+1) = c_i(t) + h(t)\frac{p_k(t) - c_i(t)}{\left\| p_k(t) - c_i(t) \right\|}$$

$$= (1,4)^T + 0.5\frac{(2\frac{2}{3},2\frac{2}{3})^T - (1,4)^T}{\left\| (2\frac{2}{3},2\frac{2}{3})^T - (1,4)^T \right\|}$$

$$= (1,4)^T + 0.5\frac{(1\frac{2}{3},-1\frac{1}{3})^T}{\sqrt{(1\frac{2}{3})^2 + (-1\frac{1}{3})^2}}$$

$$\approx (1,4)^T + 0.234 \cdot (1\frac{2}{3},-1\frac{1}{3})^T$$

$$\approx (1.390,3.688)^T$$

The other concepts are moved in a similar fashion.

It is interesting to see how this movement compares to the original ACS mechanism. The matter of interest is whether the direction is the same. The length of the movement in the space can be adjusted with the *?(t)* parameter.

For this computation of the movement in the original ACS we will treat c_0, c_1 and c_2 as terms and insert them in a string. The pairs activated are (c_0, c_1), (c_0, c_2), (c_1, c_0), (c_1, c_2), (c_2, c_0) and (c_2, c_1) in this order. The result of the associations is found in Fig. 4.5. The left grid is the same as the one in Fig. 4.4. The right grid is calculated using the association rule from the original ACS.

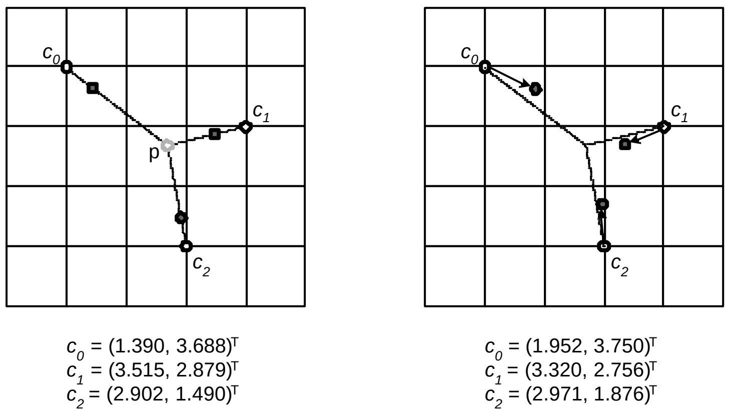


Figure 4.5 Difference between adapted learning and original learning

Figure 4.5 shows that a difference between the CASG and Schuemie's ACS exists. In the second grid c_0 has moved more towards c_1 than to c_2, and c_1 and c_2 have crept closer towards each other than towards c_0. The concepts move differently from the adapted approach because they are activated sequentially per pair.

This skewed approach is not justified by the order of appearance of the concepts in the string or weight or frequency. When weights are set at the same value for all concepts, the deviance still occurs. It is more logical to move all concepts in the string in equal measure towards each other, because all concepts in one neighborhood are of equal importance and weight. The adapted learning rule moves all concepts towards each other on an equal footing. Moving all concepts towards a central point works like activating them at the same time, and not sequentially.

4.3.2 The forgetting rule

Schuemie's forgetting rule is executed after the learning phase. When all indexes have been used as input the terms are then drawn away from each other. One by one the distance between every two terms is increased, using the forgetting rule. Two terms are moved at the same time and then the next two terms are moved based on the changed state of the distribution of terms. Because each activation of the forgetting rule uses the state of the space as the preceding activation left it, the end state after the application of the forgetting rule depends on the order of the pairs on which the forgetting rule was used.

The problem of sequentiality is the same problem as was shown for the learning rule. The order in which the forgetting rule is executed is more arbitrary than with the inverted index string, so the problem is more urgent.

For the forgetting rule a rule similar to the learning rule is suggested. The center p_S of the space S encompassing all vectors is calculated and all points are moved away from it using the rule:

$$x_i(t+1) = x_i(t) - l(\|p_S(t) - x_i(t)\|)\frac{p_S(t) - x_i(t)}{\|p_S(t) - x_i(t)\|} \qquad (4.10)$$

where $l(x)$ is the repulsion function and $l(x)$ is again defined as:

$$l(x) = \begin{cases} 1 & for\ x < 1 \\ 1/x & for\ x \geq 1 \end{cases}$$

With this forgetting rule way all concepts move away from their common center.

The rule is computationally less intensive as well. If the space contains m concepts, calculating p_S requires m vector additions and a division by a scalar. Calculating all new positions requires m applications of the forgetting rule, which consists of two vector additions and a multiplication with a scalar. In the original ACS for every combination of two terms the forgetting rule is applied. This means that the forgetting rule is used $2 \cdot \binom{m}{2} = m(m-1) \approx m^2$ times.

The forgetting rule suggested in the last section is executed after the learning stage is over. After the concepts have moved towards each other in a way that reflects their co-occurrences, they are moved away from each other depending on their distribution. It can be argued that forgetting should also reflect co-occurrence, or rather the absence of co-occurrence. The fact that only certain concepts appear together in one fingerprint goes together with the fact that others are not part of the fingerprint. In the original document certain concepts were related to a concept c_i, but a much greater set of concepts was *not* related to c_i. The author of the document did not find it necessary or interesting to mention a large set of concepts $S\backslash f_k(l)$ in the document relating to c_i. Although the fact that this does not mean the concepts in $S\backslash f_k(l)$ are not related at all, the evidence is weaker. This is the same as is reflected in the reciprocal Dice coefficient for c_i and c_j, which is lowered every time either c_i or c_j appears in a separate fingerprint.

This weakening of relations between concepts when they are not found in the same fingerprint is reflected more clearly in a selective forgetting rule. A selective forgetting rule increases the distance between concepts that are not in a fingerprint, and the concepts that are. Moving the set $S\backslash f_k(l)$ away from the center p_k of the fingerprint increases the distance to the concepts in $f_k(l)$. The distance is increased with the same forgetting rule discussed earlier.

$$x_i(t+1) = x_i(t) - l\left(\|p_k(t) - x_i(t)\|\right)\frac{p_k(t) - x_i(t)}{\|p_k(t) - x_i(t)\|} \qquad (4.11)$$

where $l(x)$ is the repulsion function and $l(x)$ is again defined as:

$$l(x) = \begin{cases} 1 & \text{for } x < 1 \\ 1/x & \text{for } x \geq 1 \end{cases}$$

After the learning rule has been applied on $f_k(l)$, the forgetting rule is used. This is different from the original ACS where the forgetting occurs after all learning in a certain epoch has finished. It increases the effect of the learning rule, because the distances within the cluster stay constant, while the distances with concepts outside the cluster are increased.

In this section a new variety of ACS has been proposed that is different from the original ACS because it activates all concepts in a fingerprints in combination with the center of gravity of the fingerprint.

4.4 Conclusion

In this chapter first the workings of the ACS algorithms have been discussed.The ACS was trained by decreasing the distance between concepts when they appear in each other's neighborhood. After the learning stage all concepts were moved away from each other again.

It was then shown how the ACS could be useful for the problems encountered in chapter 3, by making choice of path lengths possible and allowing for faster searching.

In the last section we discussed how the ACS could be improved. This improvement, the CASG will need testing to establish its value. The tests will be conducted in Chapter 5.

5 Experiments with the CASG

The previous chapter introduced the ideas behind the CASG. This, however, does not end the discussion: empirical results are necessary to test the notions put forward in Chapter 4. In this chapter several experiments will be conducted. First the principles of the ACS and the CASG are put to a test in 5.1. In the same section the right values for setting parameters are sought through experiments, as well. In 5.2 the Nature Genetics set of fingerprints is used for assessing the system on usefulness to researchers.

5.1 Tests of the principles of CASG

This section contains experiments to find out whether the choices and improvements suggested in Section 4.4 work for a test set. Before testing the CASG, methods for conducting the tests are discussed in 5.1.1. The question of whether concepts that co-occur in a test set converge in an experimental CASG or not, is treated in 5.1.2. The second question, the influence of values chosen for the various parameters, is the subject of 5.1.3.

5.1.1 Test methods

The essential quality of an ACS should be that the Euclidian distance reflects the relative frequency of co-occurrences. The tests will therefore try to establish whether this is the case for different design choices.

There are different ways to test whether distances reflect co-occurrences. The first one is to see whether concepts that co-occur frequently are located more closely in the space than concepts that do not co-occur frequently. This test, called the Closest Neighbor Test (CNT) is used to see if related concepts appear in clusters. The test consists of checking whether all concepts within a cluster are closer to each other than the smallest distance between any concept in the cluster and any concepts in another cluster.

Two clusters a and b pass this test if the largest distance between concepts within each cluster separately is smaller than the smallest distance between points c_i and c_j with $c_i \in a$ and $c_j \in b$. The workings of the CNT are illustrated in Fig. 5.1. In this figure concepts of the same cluster have the same symbol. The greatest distance between two concepts within the cluster is marked with an interrupted arrow. The smallest distance of a concept in one cluster and a concept in the other is marked with a solid arrow.

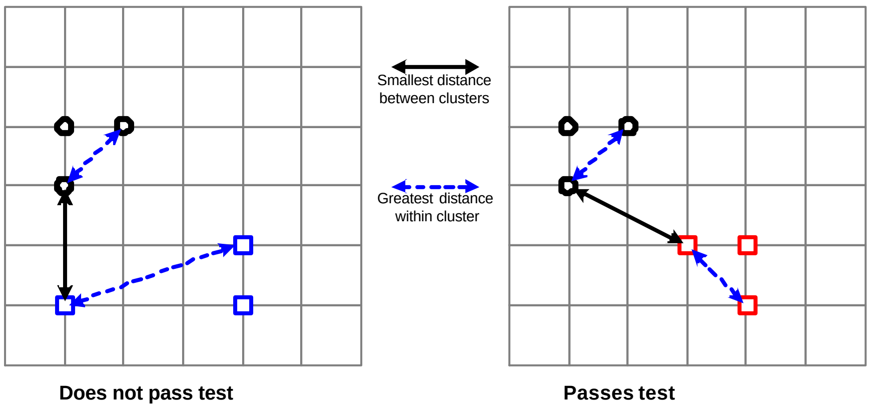

Smallest distance between clusters
Greatest distance within cluster
Does not pass test
Passes test

Figure 5.1 The Closest Neighbor test.

In Fig. 5.1 the CNT holds for the first grid and not for the second. From now on we will call the first result a positive result and the second a negative result.

The second test is to see whether the system finds paths as required, using only lines in the co-occurrence graph and with correct choices for intermediate concepts.

These tests are best first conducted with an artificially construed set of fingerprints with clear cases of co-occurrence. When the principles have been tested on clear-cut cases real fingerprint data can be used.

Before proceeding the parameters in the algorithm also need reviewing because they affect the outcome of the tests. The first parameter is the number of *epochs*. A second parameter is *?(t)*, the learning rate, the ratio at which the distance between the

concepts and p_k is reduced. The greater *?(t),* the greater the movement of the concepts in the space in case of co-occurrence. A third parameter is the forgetting rate *?(t)* which determines how strong non-co-occurring concepts are pushed away from p_k. This parameter is set in the same way as Schuemie proposed as was described in Chapter 4. A fourth parameter is the number of dimensions used in the CASG. The last parameters are the bounds on the initial random assignments of the concepts. The concept vectors are assigned *n* random numbers as an initial location. These *n* random numbers are generated using a pseudo random number generator that produces numbers in the interval [*a*,*b*]. These bounds *a* and *b* have to be set.
All tests were run on a Pentium III running at a clock speed of 1 GHz with 256 MB of memory. The programming was done in Microsoft Visual C++ 6.0. The results are reported in the form of charts. The number on which the graphs are based are found in the tables in appendix A.

5.1.2 Test of principles

The CASG is intended as an algorithm to cluster related concepts by assigning them positions close to each other. An important test is therefore whether the clustering does indeed occur.
For this purpose a test set is constructed, out of artificial conceptual fingerprints. The set consists of 30 fingerprints each containing 10 concepts.

$$\begin{bmatrix} 0 & 1.0 \\ 1 & 1.0 \\ \vdots & \vdots \\ 8 & 1.0 \\ 9 & 1.0 \end{bmatrix}_1, \begin{bmatrix} 10 & 1.0 \\ 11 & 1.0 \\ \vdots & \vdots \\ 18 & 1.0 \\ 19 & 1.0 \end{bmatrix}_2, \ldots, \begin{bmatrix} 280 & 1.0 \\ 281 & 1.0 \\ \vdots & \vdots \\ 288 & 1.0 \\ 289 & 1.0 \end{bmatrix}_{29}, \begin{bmatrix} 290 & 1.0 \\ 291 & 1.0 \\ \vdots & \vdots \\ 298 & 1.0 \\ 299 & 1.0 \end{bmatrix}_{30} \qquad \text{(Test Set 1)}$$

This set contains 300 different concepts, all with weight 1.0. In this set the concepts numbered 0 to 9 all co-occur with each other, as do the concepts 10 to 19 and so on until 299.
For the test we will focus on the 30 clusters each containing all concepts of one fingerprint. Because these fingerprints show no overlap the fingerprints should be completely separated. This means that CNT should give a negative outcome for all different combinations of clusters. The next table lists the average number and

standard deviation s of pairs of positively tested clusters in 10 runs after *m* epochs. In this first test *?(t)* = 2 and the *d* as specified in chapter 4. The bounds a = 0, b = 50. The dimension is *n*=9.

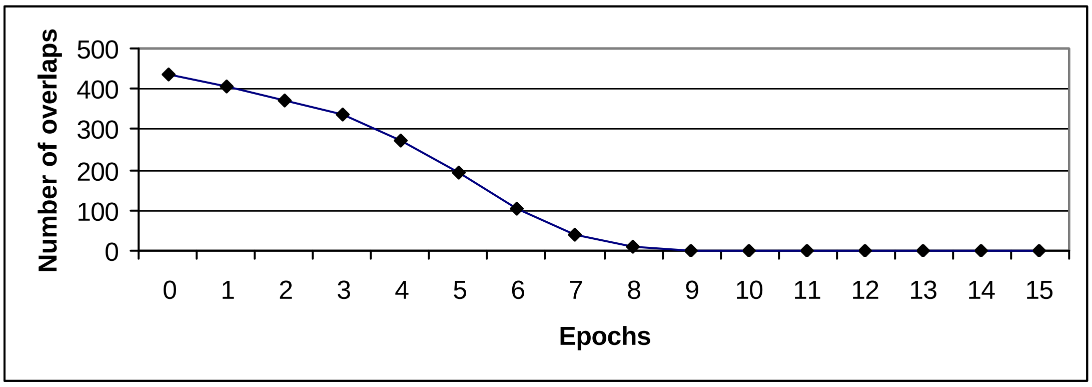


Figure 5.2 The average number of positive test against the number of epochs for independent fingerprints (table A.1)

The most important conclusion from Fig. 5.2 is that all clusters are separated after at most 10 epochs, which is the desired result. Without training the CNT is positive for pairs of all clusters. The standard deviation *s* equals 0 for the untrained CASG. The rates of improvement for the different runs diverge from here and the s increases. It decreases again after the epoch 5 as the rates converge again.

The next test is whether the CASG handles overlapping articles correctly. As a test set the following fingerprints will be used:

$$\begin{bmatrix} 0 & 1.0 \\ 1 & 1.0 \\ \vdots & \vdots \\ 18 & 1.0 \\ 19 & 1.0 \end{bmatrix}_1, \begin{bmatrix} 10 & 1.0 \\ 11 & 1.0 \\ \vdots & \vdots \\ 28 & 1.0 \\ 29 & 1.0 \end{bmatrix}_2, \ldots, \begin{bmatrix} 280 & 1.0 \\ 281 & 1.0 \\ \vdots & \vdots \\ 298 & 1.0 \\ 299 & 1.0 \end{bmatrix}_1, \begin{bmatrix} 300 & 1.0 \\ 301 & 1.0 \\ \vdots & \vdots \\ 318 & 1.0 \\ 319 & 1.0 \end{bmatrix}_{30} \qquad \text{(Test Set 2)}$$

We are first interested in the clustering of these concepts. This must be tested for sets of 10 concepts that should grouped together. The average is taken over ten experiments, using the same parameters as previously.

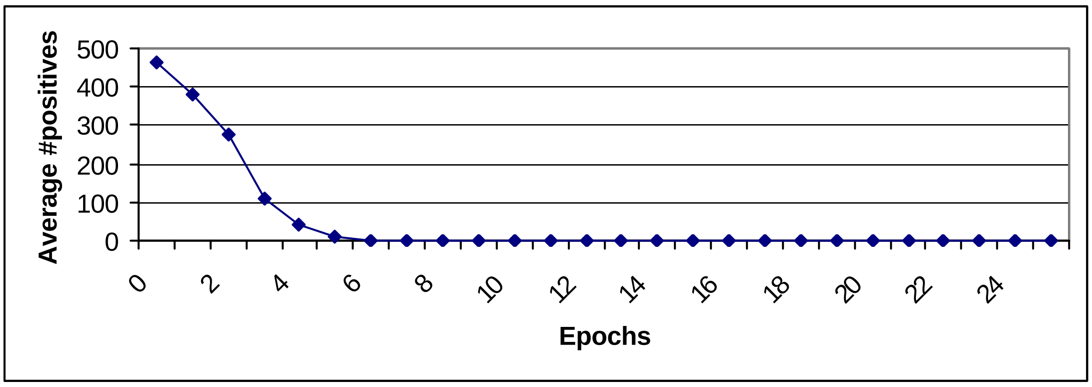


Figure 5.3 The average number of positives of 10 concepts against the number of epochs for overlapping fingerprints (table A.2)

For groupings of 10 related concepts the results in Fig. 5.3 look good. To separate the clusters even fewer epochs are needed. This is because each cluster of 10 concepts is found in 2 fingerprints instead of 1 as in Test Set 1. In some case the first and the last cluster of 10 are hard to separate, because for this group half of the material is available.

Now that clustering seems possible for both test sets, the finding of paths in this hypothetical test set is of interest, especially for Test Set 2.

In Test Set 2 a path is possible between any two given concepts, because the articles contain overlap, using the same parameters as previously.

A path from 0 to 25 looks like 0-13-25. This is what we expect. 0 co-occurs with 13, and 13 with 25. The fact that 13 is chosen and not any other concept from the range [10,19], is not a consequence of the fingerprint, but merely of the initial distribution of the concepts. All concepts in the range [10,19] co-occur as frequently with 0 as thirteen, but one is selected using edge length. The edges have slightly different edge lengths due to the initial random distribution. On the basis of these slightly different edge lengths 13 is selected.

The longest path in the CASG based on Test Set 2 is from a concept of the range [0,9] to a concept in the range [300,309]. This has been tested for a path between 0 and 309. The path found using A* is:

0-13-25-39-43-52-69-72-83-96-102-117-122-133-148-150-163-179-188-192-209-212-223-239-245-253-262-273-289-298-309

As you can see every bracket of 10 concepts is part of the path, and no node in the path could have been excluded.

The conclusion of the test of principles is that for an artificial set the clustering takes place as required. Concepts that co-occur cluster together and concepts that co-occur frequently cluster closer around the concept. Using the CASG paths can be found between concepts that meet requirements as well.

5.1.3 Setting the parameters

As with many AI algorithms the choice of the parameters is very important and it is difficult to theorize about their values.

A first test compares the differences between two CASG of the same test set (Test Set 1) using a constant and a variable learning rate. The numbers in the table are obtained in the same way as the ones in the total column in Fig. 5.2.

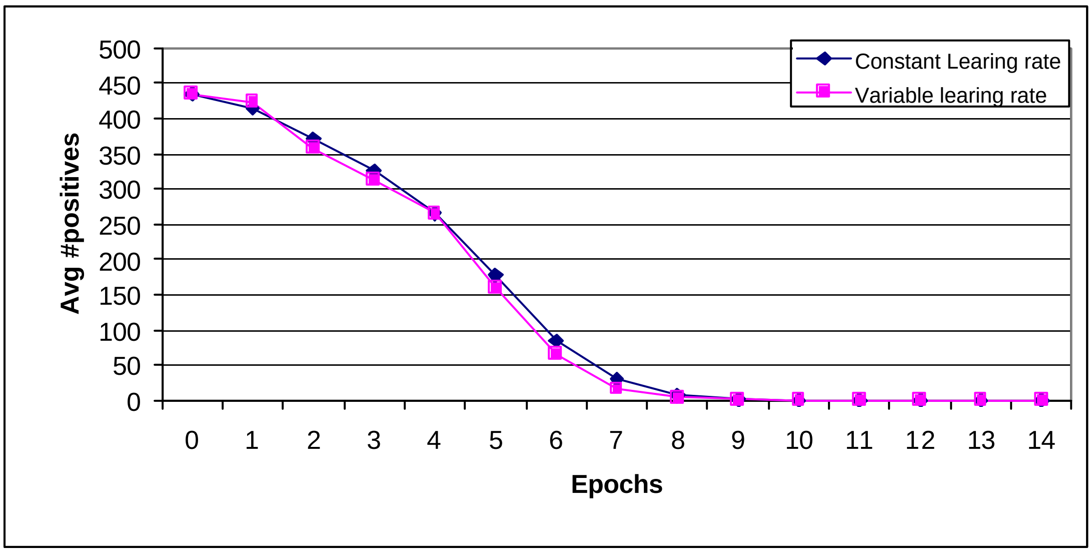


Figure 5.4 Difference between variable and constant learning rate.

The choice of one or another learning rate seems to make little difference in the CNT. In these experiment of 10 runs the variable learning rate seems slightly more efficient.

Another parameter tested here will be the choice of *n*, the number of dimensions. The idea behind the choice of *n* is the following: The higher *n*, the higher the number of degrees of freedom in the positioning of concepts. Increasing n can thus improve the location. This is tested for n = 3, n = 9 and n = 15.

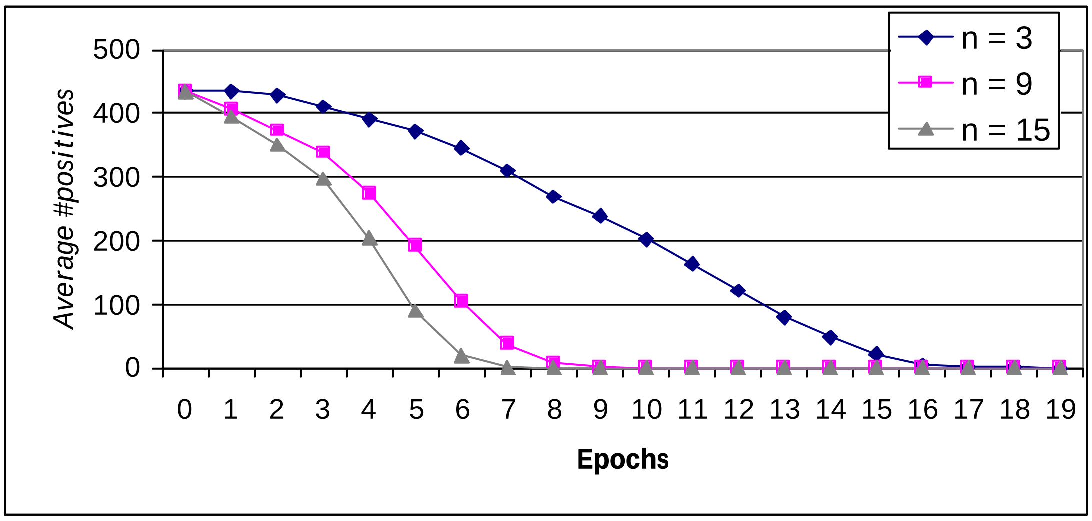


Figure 5.5 The average number of positive test for different dimension

The experiments show that the CASG learns in fewer epochs when the n is increased. The size of the increase decreases in n. In other words, the difference between n = 3 and n = 9 is larger than between n = 9 and n = 15.

There are more parameters to be tested. The forgetting rule was defined as:

$$x_i(t+1) = x_i(t) - l\left(\left\|p_k(t) - x_i(t)\right\|\right)\frac{p_k(t) - x_i(t)}{\left\|p_k(t) - x_i(t)\right\|}$$

where $l(x)$ is the repulsion function and $l(x)$ is again defined as:

$$l(x) = \begin{cases} 1 & for\ x < 1 \\ 1/x & for\ x \geq 1 \end{cases}$$

By changing the repulsion function to:

$$l(x) = \begin{cases} 1 & for\ x < q \\ 1/x & for\ x \geq q \end{cases}$$

and varying *q* forgetting behaves differently. The effect is most interesting on overlapping sets. Therefore Test Set 2 is used.

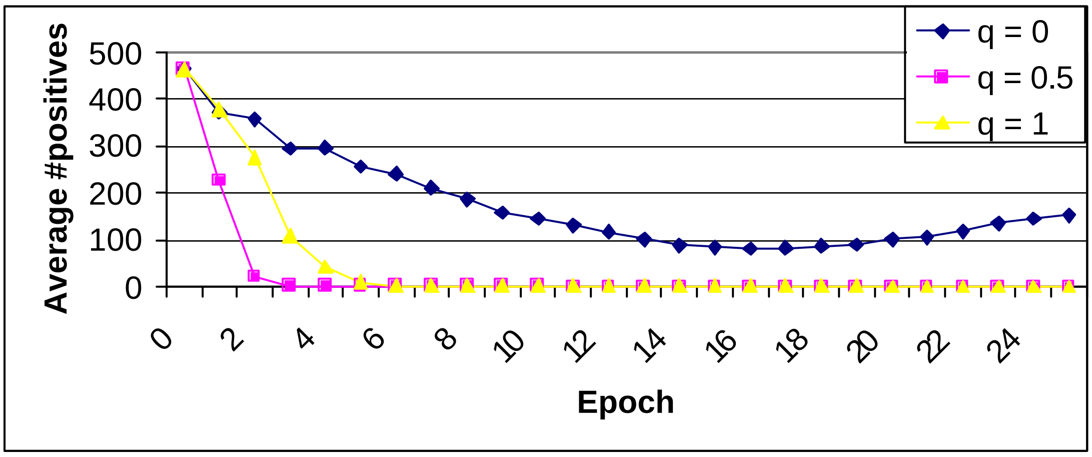


Figure 5.6 The average number of positive tests for different forgetting rates

Setting *q* = 0.5 yields better results and allows faster separation of the clusters. Lowering to *q* = 0 works badly, because the effect of the forgetting rule on two concepts close to each other is so strong it counters the learning rule after some time
The experience with the set allows us to work with the Nature Genetics data set. We have seen that there is little difference between a variable and a constant learning rule. It also became clear that increasing the number of dimensions leads to a smaller number of epochs required to train the CASG. The size of the increased efficacy decreases with the number of dimensions added. The last test involved the forgetting rule. Our experiments suggest that setting *q* = 0.5 is a good idea.

5.2 Using a real data set

Using a real data set in a CASG is quite different from the use of artificial test sets. A set of real fingerprints is much more irregular in its distribution and placement. Testing is also different because the expected results are not very clear. There are two kinds of test we can run. The first kind is the mechanical test, which is used to see whether the CASG has the properties required. These tests will be explained in 5.2.1. The second kind of test, described in 5.2.2, is designed to see whether the paths are meaningful or not. Experts are needed for these tests.

5.2.1 The properties of CASG based on real sets

In the previous section tests were conducted on how the various learning parameters influence the properties of the CASG. In this section the focus will be on how the CASG behaves when used for a larger data sets, derived from Nature Genetics articles.

A test is whether the CASG stabilizes in time. Looking at the total movement of the concepts in the CASG in every epoch is a test for stability. The expectation is that much movement is seen initially, after which the concepts have found their place in more or less stable locations.

The total amount of movement looks like this:

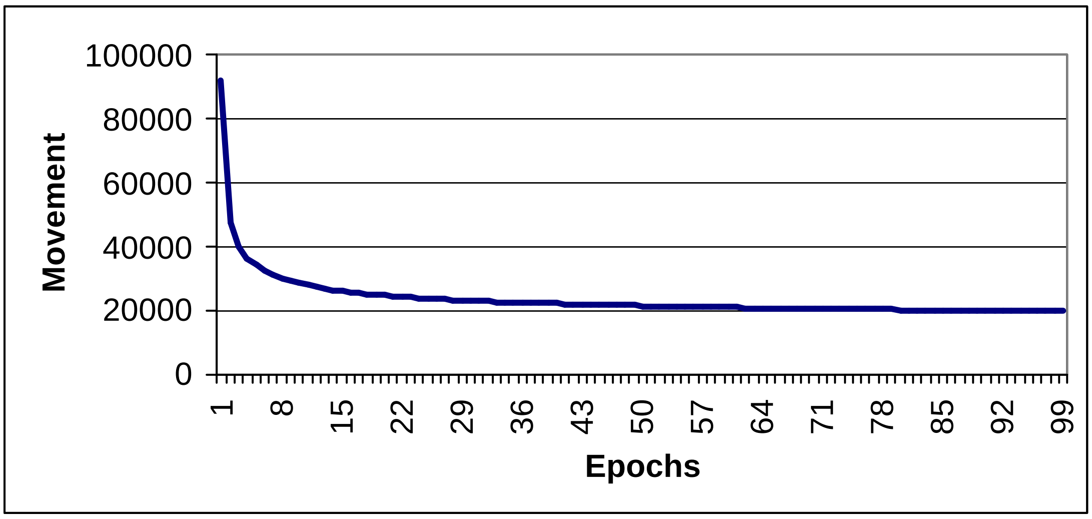


Figure 5.7 The size of the movement in space

The test was done with standard settings: a constant learning rate, forgetting at $q = 1$, dimension $n = 9$. Figure 5.7 shows that the initial movement is very large and quickly decreases. The decrease in movement becomes very small in the end, although the amount of movement never stops declining. The result is just as hoped for.

5.2.2 Use of paths for hypothesis testing

The last question that needs answering is whether the CASG gives paths in the Nature Genetics set that users find interesting. This property is very difficult to measure. Specificity and recall are no use because then there has to be a result expected by experts. The point of the system, however, is to generate unknown links.

It is also difficult to take random concepts and generate a path between them, because the concepts have to be contained in the graph and they are only of interest when no path of one edge exists.

An ideal test would require a panel of experts that would think up hypotheses and test them in the literature. A first, simpler, test is to take paths between concepts selected from the CASG and show them to a medically trained person. This is what has been done as a second test of the system.

A first test is to see whether the clustering seems logical. This will be tested not in a systematic way, but will give an idea of the usefulness of the system.

"Telomerase" closest neighbor is "Telomere", which is logical from a biomedical perspective. The next closest neighbors are in order of proximity: "Universities", "Administration" and "Cultural". What these have to do with telomerase is not clear. For "Breast Cancer" the closest neighbors are "Risk", "cDNA microarrays", "Libraries", "Nucleus Accumbens" and "Increased body temperature". Of this series risk and cDNA microarrays make most sense directly.

As a last illustration: the concept "genes". Its neighbors are: "Bacteria", "Trees", "Bacterial", "Calculi", "Escherichia coli", "Workforce".

The results of this first test are mixed. Some interesting concepts are found near a seed concept. It also happens with all three tested that very general concepts show up, whose information value is at least dubious.

The second test involves finding paths. To illustrate this process we have searched for three paths between informative concepts in the graph. Unfortunately they are not representative because these are not paths experts will look at.

I.

Find a path between "Telomerase" and "Breast cancer". The path found between these concepts goes via "Mutation". This makes sense, because in telomeres in the DNA mutations take place and DNA mutations cause breast cancer.

II.

Find a path between "Sialic acids" and "Multiple sclerosis". The path found contains first "Cells" and then "Exons".

III.

Find a path between “Telomerase” and “Multiple sclerosis”. The path found contains “Telomere” and “Population”. The term “Population” is very general andprobably does little to clarify a relation between “Telomerase” and “Multiple sclerosis”.

These tests must be improved, because one hardly expects to find a relation between sialic acids and multiple sclerosis, as is tested in the previous paragraph. A better test would use real hypotheses, but such test were not conducted within this thesis.
The first path is interesting, but the other two seem very general. The approach is promising, but more refined tests of the network are needed to establish usefulness of the system.

5.3 Conclusion

The past chapter has shown that the system behaves as the CASG is expected to do. It separates clusters when it has to. It creates paths the way it should. However, testing these paths to see whether they are interesting is difficult. Nevertheless, the results look promising. These results will be put in perspective in the next chapter.

6. Discussion and outlook

This final chapter gives an overview of the results of this thesis. These results are assessed in 6.1. In 6.2 suggestions are made for research that can further develop the system suggested in this thesis. In 6.3 the thesis will be concluded.

6.1 Assessment

The CASG has been gradually set out in the previous chapters. First we will discuss what has been done, then the value of what has been done and, finally what the limitations are. In the next section suggestions for further research are made.

6.1.1 Requirements

In Chapter 5 we have tested the CASG for both an artificial set and a real set of fingerprints. The results for the test of the artificial set were good. Both the testing and interpreting of the real data set were more difficult.

To see what the value of the system is, the goals from Chapters 1 and 2 are needed.

The main goal was to find knowledge paths in scientific literature. In this thesis an approach to that goal has been presented. Concepts in the literature are represented as nodes in a graph. Between co-occurring concepts edges exist. Every node has a location in an *n*-dimensional space. The distance between two nodes that can be calculated on the basis of these locations reflects their relatedness. Paths between concepts in the space, using edges, can be constructed. The resulting paths reflect knowledge of relations in the literature.

The main goal was therefore reached, although the system is still in an experimental phase and needs further refinements before it will have practical use.

For now the main limitation of the system is its inability to distinguish between informative and non-informative concepts. Quite a few paths found during testing were very general and at a first glance not very interesting. Use of filtering and adapting of the training rules, as proposed in 6.2 might help to overcome this problem.

Apart from the main goal, it was also deemed desirable that:

- *The system has to be able to generate paths within reasonable time, which means minutes rather than days.*
  Searching for a path takes a few seconds at most, so this condition is fulfilled.
- *The system has to require little specific domain knowledge from the user.*
  The user needs domain knowledge only to interpret the results, not to define his query.
- *The system has to be able to deal with a large number of articles.*
  The tests were done for the Nature Genetics sets, which contained 708 for the tests in Chapter 4 and 944 fingerprints for the tests in Chapter 5. These numbers were handled within reasonable times. Training took a few hours.

The CASG complies with the conditions. The question that remains is whether the paths found are of any interest for researchers for whom it is meant. The first test show that sometimes they are but sometimes they are not.

6.2 Suggestions for further research

Much of what is described in this thesis is experimental in the sense that it still needs further development. Chapter 5 and Section 6.1 showed that the results are still far from practical. In this section suggestions will be made for the further improvement of the system. This is subdivided in four subsections: 6.2.1 deals with issues of increasing the amount of information derived from articles, 6.2.2 deals with improvements in the use of domain knowledge, 6.2.3 deals with possible improvements in the algorithm and 6.2.4 is about testing issues.

6.2.1 Increasing information use of data set

Conceptual fingerprints are the source of information used to train the CASG. All concepts are attracted to each other when they appear jointly in a fingerprint. The way this information is used is fairly straightforward. Some adaptations might improve the amount of knowledge derived from fingerprints.

One improvement would deal with the frequency of concepts. Because in the Genetics test set some concepts, e.g. "genes", occurred extremely frequently, they are uninformative within the test set. Schuemie proposed weighing to deal with this issue. He used as a multiplication factor:

$$a(i,j) = \frac{2\,freq_*}{freq_i + freq_j} \qquad (6.1)$$

and where $freq_i$ is the frequency of term i in the string of all terms and $freq_*$ is the average frequency of a single term.

This rule is easily adapted to work with concepts, and might improve the distribution of the CASG further. The weight decreases the importance of co-occurrence with concepts with an above average frequency and increases importance of co-occurrence with concepts of under average frequency in the test set.

A second improvement deals with the order information available in some fingerprint formats. In this thesis, concepts ordered by relevance were used which means that the order of the concepts in the original article is lost. This order information is, however, available in other formats, where locations of concepts in the original articles are stored. This format resembles the book indexes used by Schuemie. In this thesis a certain level of aggregation was chosen for co-occurrence. All combinations of concepts found in one article were used. It is nevertheless possible to decrease the window size to focus on paragraphs instead of entire articles. The choice of aggregation level depends on one's requirements.

A third improvement would make greater use of the relevance score. In this thesis the relevance score was used only for cutting off. However, the score can also be brought into action as a weighting for the learning rule or in the calculation of p_k. Highly relevant concepts could be attracted to p_k in greater measure than weakly relevant ones. The other option is that highly relevant concepts play a greater role in the calculation of p_k. This could be by the following rule:

$$p_k = \begin{bmatrix} p_{k,1} \\ \vdots \\ p_{k,n} \end{bmatrix} = \begin{bmatrix} \dfrac{\sum_{l=0}^{m} r_{i,k} c_{i,1}}{\sum_{l=0}^{m} r_{i,k}} \\ \vdots \\ \dfrac{\sum_{l=0}^{m} r_{i,k} c_{j,n}}{\sum_{l=0}^{m} r_{i,k}} \end{bmatrix} \tag{6.2}$$

These improvements seem easy to implement and might improve the accuracy of the CASG.

6.2.2 Use of domain knowledge

In the previous subsection the use of information in fingerprints was discussed. In this subsection additional sources of information are proposed. These sources are the semantic network and the hierarchical tree of the UMLS.

All concepts found in fingerprints have one or more semantic categories attached to them in the UMLS. These categories give the concepts a place in a semantic network. Every concept has an hierarchical relation attached to it as well. This means that each concept has descendants and predecessors, where predecessors are more general concepts and descendants more specific. This information might be exploited in the training of the CASG or its interpretation.

In 3.1.4 it was argued that a co-occurrence measure that used child co-occurrence, or descendant co-occurrence would be better. In the implementation of the CASG they were not incorporated, but it would certainly be possible to exploit hierarchical relations in the training.

One idea would be to use an activation that spreads over family relations. This would mean that, first, all concepts c_i in an article are attracted to p_k and then all children of c_i are attracted to p_k, albeit to a lesser extent, and then their children, etc. This might lead to exponential time use, so some cut-off would be needed, e.g. after two generations.

The amount of information stored in the semantic network of UMLS is low. The network defines possible relations between semantic categories and does not store whether they are actualized. This information can however be useful when interpreting paths in the CASG.

In the CASG edges stand for co-occurrence relationships. An edge has no other meaning than that the two concepts it connects appear at least once together in one fingerprint. No semantics are attached to the relationship. In the original texts a semantic relation did exist. One way to find the semantics of the relation would be to use NLP techniques to locate the verbs that connect the concepts. Another is using the semantic network. Between two semantic categories a limited number of semantic relations exist.

Each concept also has a limited set of semantic categories. This means that the set of possible semantic meanings of the relation between two concepts can be narrowed down using the semantic network. Unfortunately the exact nature of the relation can only very rarely be pinpointed because most concepts fall under several semantic categories and between most categories several relations exist.

6.2.3 Improvements of the algorithm

In the previous two subsections the use of information sources was discussed. In this subsection the focus will be on how dealing with information might be improved. In the path search so far no penalty for the number of stops has been added. Adding a penalty would force the A* algorithm to choose paths consisting of fewer concepts.

Another improvement would allow the user to ban certain concepts or categories of concepts. Certain very general concepts are hardly of interest to the user. One way to deal with those concepts is to prevent the algorithm from selecting them.

6.2.4 Testing methods

One problem encountered during the final phases of the thesis is lack of standard testing methods for the value of paths. Good tests are essential for the further development of the method. A test panel of experts could look at certain paths between concepts they know well and judge whether they think the paths are of interest. It will be difficult to develop an objective test, because of the many possible answers of a query.

6.3 Overall Conclusion

Although the system proposed in this thesis evidently needs further research before it can become fully operational, the central notions seem to work. Some of these ideas are new. The first new idea was to use fingerprints instead of articles or parts of articles as a basis for co-occurrence. The second new idea was to use an *n*-dimensional space to give the nodes of a co-occurrence graph a location, which helps to deal with searching and assigning weights to edges. This way the advantages of a graph, clearly defined paths, and of a conceptual space, distances between every pair of concepts, are combined.

The system has limitations, most of which already have been mentioned in 6.1 and 6.2, but looks promising. Further research is necessary to further refine this interesting path.

APPENDIX A RESULTS

These are the results of different test conducted within the framework of Chapter 5. For their meaning please refer to that chapter.

Epoch	Avg #positives	s^2
0	435	0
1	406.8	13.34832
2	373	13.06395
3	338	13.5072
4	274.5	35.44714
5	192.1	44.01628
6	104.9	33.77195
7	37.9	16.10003
8	9.1	3.928528
9	1.4	0.843274
10	0	0
11	0	0

Table A.1 The average number of positive test against the number of epochs for independent fingerprints

Epoch	Avg #positives	s^2
0	465	0
1	379.875	15.05644
2	276.5	101.7057
3	108.8	129.169
4	43.3	81.81694
5	10.1	19.20908
6	1.7	1.159502
7	1.3	0.674949
8	1.4	0.699206
9	1	0.666667

10	0.9	0.567646
11	0.5	0.527046
12	0.7	0.674949
13	0.9	0.737865
14	0.9	0.737865
15	1.1	0.737865
16	0.9	0.737865
17	0.7	0.823273
18	0.4	0.516398
19	0.4	0.699206
20	0.2	0.421637
21	0.2	0.421637
22	0.2	0.421637
23	0.1	0.316228
24	0	0
25	0	0

Table A.2 The average number of positives of 10 concepts against the number of epochs for overlapping fingerprints

Epoch	Avg #positives	s^2
0	435	0
1	414.7	13.76832
2	370.6	13.14196
3	326.4	21.04598
4	265.7	14.62912
5	176.6	23.54759
6	84.5	23.94554
7	28.9	11.62803
8	6.2	3.259175
9	0.8	0.788811
10	0.1	0.316228
11	0	0
12	0	0
13	0	0
14	0	0

Table A.3 Difference between variable and constant learning rate

	n = 3		n = 15	
Epoch	Avg #positives	s^2	Avg #positives	s^2
0	435	0	435	0
1	434.9	0.316228	395.3	11.60508
2	428	8.956686	351.4	20.64623
3	411.1	19.43336	297.4	23.99167
4	391.7	29.4658	204.7	31.73869
5	371.8	35.452	90.7	30.86188
6	345.9	46.16023	20.1	11.53208

7	310.2	51.37834	2	1.563472
8	269.5	61.69324	0.1	0.316228
9	238.9	70.221	0	0
10	202.6	74.59699	0	0
11	163.7	70.64316	0	0
12	121.4	59.30561	0	0
13	80.5	48.92909	0	0
14	48.2	38.58267	0	0
15	22	20.07209	0	0
16	5.7	6.532823	0	0
17	1.9	2.18327	0	0
18	0.8	1.316561	0	0
19	0.6	0.966092	0	0

Table A.4 The average number of positive test for different dimension

	q = 0	q = 0.5
Epoch	Avg #positives	Avg #positives
0	465	465
1	373.8	226.8
2	357.2	20.9
3	294.5	1.3
4	295.9	1.5
5	256.7	1.1
6	241.3	0.7
7	211.1	0.7
8	186.6	0.8
9	158.8	0.8
10	145.9	0.6
11	130.5	0.3
12	116.8	0.2
13	101.3	0.1
14	88.7	0
15	84.7	0
16	81.7	0
17	82.7	0
18	86.7	0
19	90.3	0
20	101.3	0
21	105.7	0
22	119.3	0
23	135.8	0
24	145.7	0
25	153.3	0

Table A.5 The average number of positive tests for different forgetting rates

APPENDIX B CODE

The code is all written in Microsoft Visual C++ 6.0. The most relevant parts are listed here.

First there are the data structures used for the implementation. The structures are vector and graph. The vector implementation was copied and adapted from an implementation by Jeroen Donkers of Maastricht University. The files with this data type plus more functionality not used in this thesis are found in the files mattype.h, rows.h, row.cpp, matrix.h and matrix.cpp. This data structure will not be listed.

The graph was implemented as is explained in (19). The difference is that one can store information in each node and on each edge. The implementation is found in the file graph5.h and graph5.cpp. Because the principles are clearly described in Decker this data structure will not be listed.

The sets of fingerprints from Collexis are loaded into a Standard Template Library (STL) data structure called a Map. A map is a template collection of pairs of a key and data element. The Map that contains the data has the unique article identifier for a key and the fingerprint as the data element. The fingerprint is of the STL-type vector, which is an array. Each array element is a pair of a concept identifier (a long integer) and relevance score (a double float). This is illustrated in Fig B.1.

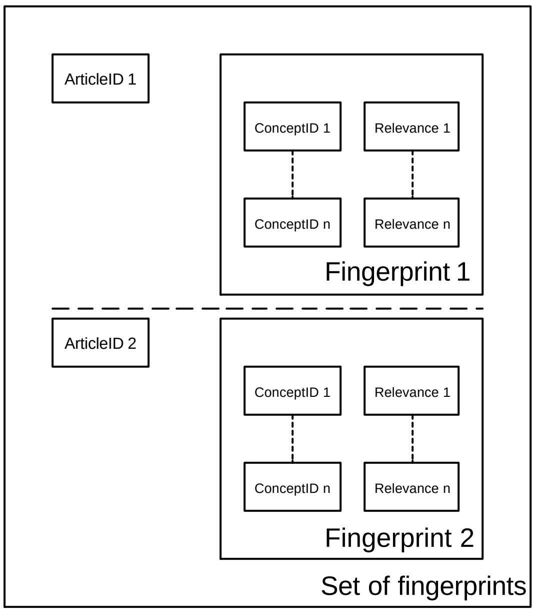


Figure B.1 The structure of the set of fingerprints

The Fibonacci heap was implemented using (21) and (23) and the LEDA implementation of this data structure (www.algorithmic-solutions.com/as_html/products/leda/products_leda.html). It will not be listed here.
The listings of two data structures will be given here: SpatialGraph and ConceptTrainer. SpatialGraph is a graph with a vector assigned to each node. The graph has a number of member functions. One important one is the A* algorithm for finding paths between two concepts in the graph. The length of the paths is equal to the total of Euclidian distance of each of the edges it consists of.
ConceptTrainer is the actual CASG. It contains functionality for training and forgetting articles.

The most important feature of SpatialGraph is the A* implementation and the member functions to be able to execute A*.

```
//------------------------------SpatialGraph.h--------------------------------------
#include "graph5.h"
#include "graph5.cpp"
#include "fibonacciheap.h"
#include "fibonacciheap.cpp"
#include "matrix.h"
#include <map>
#include <limits>
#include <list>
```

```
const REAL infinity = numeric_limits<REAL>::infinity();

template<class M,class AttrClass>
class SpatialGraph : public Multigraph<M,Vector,AttrClass>
{
	public:
		list<M> HeuristicOptimalPath(const M& source, const M& target);
		REAL PathWeight(list<M> path);
		REAL Distance(const M& source, const M& target);
		list<M> Neighborhood(const M& center, REAL diameter);
	private:
		REAL EstimatePathWeight(const M& source, const M& target);
};

template<class M>
class EstimateMap : public map<M,REAL>
{
	public:
		REAL get_value(const M& object);
};

template<class M>
class AstarNode
{
	public:
		AstarNode(const M& pname, const  REAL& pg_estimate)
			{name = pname; g_estimate = pg_estimate;};
		void SetG(const REAL& pg_estimate) {g_estimate = pg_estimate;};
		REAL GetG() const {return g_estimate;};
		void SetName(const M& pname) { name = pname; };
		void SetParent(const M& pparent) {parent = pparent;};
		M GetParent() const {return parent;};
		M GetName() const {return name;};
	private:
		M name;
		REAL g_estimate;
		M parent;
};

template<class M>
bool operator== (const AstarNode<M>& node1,const AstarNode<M>& node2) {return node1.GetName() ==
node2.GetName();};
template<class M>
bool operator!= (const AstarNode<M>& node1,const AstarNode<M>& node2) {return node1.GetName() !=
node2.GetName();};
template<class M>
bool operator< (const AstarNode<M>& node1,const AstarNode<M>& node2) {return node1.GetName() <
node2.GetName();};


//-------------------------------SpacialGraph.cpp---------------------------------------------

#include "SpacialGraph.h"
#include <assert.h>
#include <set>


template<class M> REAL EstimateMap<M>::get_value(const M& object)
{
	map<M,REAL>::iterator itobject = map<M,REAL>::find(const M& object);
```

```
        REAL result = infinity;
        if(itobject == end())
                result = itobject->second;

        return result;
}

template<class M, class AttrClass> REAL SpacialGraph<M,AttrClass>::PathWeight(list<M> path)
{
        REAL result = 0;
        list<M>::iterator lastnode = path.begin();
        for(list<M>::iterator currentnode = path.begin(); currentnode != path.end(); ++currentnode)
        {
                assert(Contains(*currentnode));
                result += (Retrieve(*currentnode)-Retrieve(*lastnode)).L2();
                lastnode = currentnode;

        }
        return result;
}

template<class M, class AttrClass> REAL SpacialGraph<M,AttrClass>::Distance(const M& source, const M&
target)
{
        REAL result = infinity;
        if(Adjacent(source,target))
                result = (Retrieve(source) - Retrieve(target)).L2();
        return result;
}

template<class M, class AttrClass> REAL SpacialGraph<M,AttrClass>::EstimatePathWeight(const M& source,
const M& target)
{
        assert(ContainsVertex(source) && ContainsVertex(target));
        return (Retrieve(source)-Retrieve(target)).L2();
}

template<class M, class AttrClass> list<M> SpacialGraph<M,AttrClass>::HeuristicOptimalPath(const M& source,
const M& target)
{
        assert(ContainsVertex(source) && ContainsVertex(target));
        list<M> result;
        list<M> neighbours;
        FibonacciHeap<AstarNode<M>,REAL> S_tilde;
        set<AstarNode<M> > S;
        REAL distancekx;
        REAL f_temp;

        bool found = false;
        list<M>::iterator x;
        set<AstarNode<M> >::iterator itS;
        AstarNode<M> first(source,0);

        S_tilde.Insert(first, EstimatePathWeight(source,target));

        while(S_tilde.Size()!=0 && !found)
        {
                FibonacciNode<AstarNode<M>,REAL> k = S_tilde.PopMin();
                cout.flush();
                if(k.GetName().GetName() == target)
}           found = true;
        else
        {
            neighbours = Neighbours( k.GetName().GetName() );
            int i = 0;
            for(x = neighbours.begin(); x != neighbours.end(); ++x)
            {
                distancekx = Distance(k.GetName().GetName(), *x);
                AstarNode<M> xnode( *x, k.GetName().GetG() + distancekx);
                xnode.SetParent(k.GetName().GetName());
                f_temp = xnode.GetG() + EstimatePathWeight(*x,target);

                if( S_tilde.Contains(xnode) )
                {
                    if(S_tilde.GetName(xnode).GetG() > xnode.GetG())
                    {
                        // Not too elegant, could be improved
                        f_temp = S_tilde.GetKey(xnode);
                        S_tilde.Delete(xnode);
                        S_tilde.Insert(xnode, f_temp);
                    }
                }
                else
                {
                    itS = S.find(xnode);
                    bool test = (itS == S.end());
                    if(itS != S.end())
                    {
                        if( itS->GetG() > xnode.GetG() )
                        {
                            S.erase(itS);
                            S_tilde.Insert(xnode, f_temp);
                        }
                    }
                    else
                    {
                        S_tilde.Insert(xnode, f_temp);
                    }
                }
            }
        }
        S.insert(k.GetName());
    }

    if(found)
    {
        // Construct path
        AstarNode<M> current(target, 0);
        M name = target;
        do
        {
            current.SetName(name);
            itS = S.find(current);
            result.push_front( itS->GetName() );
            name = itS->GetParent();
        }
        while(current.GetName() != source);
    }
    return result;
}
```

```
template<class M, class AttrClass> list<M> SpacialGraph<M,AttrClass>::Neighborhood(const M& center, REAL
diameter)
{
        assert(ContainsVertex(center));
        list<M> result;

        // Traverse the vertex list
        VertexCell<M,Vector,AttrClass>* vPtr = gHead;
        VertexCell<M,Vector,AttrClass>* centerPtr = Find(center);
        Vector location = centerPtr->vert->GetData();
        while(vPtr)
        {
                if( (vPtr->vert->GetData()- location).L2() <= diameter )
                        result.push_back(vPtr->vert->GetID());
                vPtr = vPtr->next;
        }
        return result;
}
```

ConceptSpace is the complete CASG. It contains a SpatialGraph and stores the necessary parameters. It implements the training of the CASG.

```
//-------------------------------------ConceptSpace.h--------------------------
#include "fileread.h"
#include "SpacialGraph.h"
#include "SpacialGraph.cpp"
#include "spanbox.h"
#include "matrix.h"
#include <stdlib.h>             // for rand
#include <time.h>

typedef map<long, Vector> Vectormap;
typedef SpacialGraph<long, int> Cooccurrencegraph;

REAL* RandomArray(int size, REAL lowbound, REAL upbound);
// returns an array of random REALS (defined in mattype.h) between lowbound
// and upbound

Vector CentralPoint(Vectormap pointslist);
// Return the point in the centre of all points

list<long> ListofFirst(ValueVector input);

class ConceptSpace
{
public:
        ConceptSpace(int pdimension);
        void ChangeSpace(const Cooccurrencegraph& graph) {cooccurrencegraph = graph;};
        void SetCutoff(float pcutoff) {cutoff= pcutoff;};
        void SetBounds(REAL plowerbound, REAL pupperbound)
                { lowerbound = plowerbound; upperbound = pupperbound;};
        void SetLearningrate(float plearningrate)
                { assert(plearningrate >= 0); learningrate = plearningrate;};
        void SetMinlearningRate(float pminlearningrate)
                { assert(pminlearningrate >= 0); minlearningrate = pminlearningrate;};
```

```
	void Initialize(SimpleVectorMap articleset);
	REAL Distance(long a, long b);
	Cooccurrencegraph GetCooccurrencegraph() {return cooccurrencegraph;};
	REAL LearnArticle(ValueVector fingerprint);
	void Train(SimpleVectorMap articleset, int numepochs);
	void SetCooccurrencegraph(const Cooccurrencegraph& a) {cooccurrencegraph.Clear();
cooccurrencegraph = a;};
	SpanBox ClusterBox(list<long> conceptlist);
protected:
	REAL Attract(const Vector& pcentralpoint, long id, float learningrate);
	REAL Deter(const Vector& pcentralpoint, long id, float forgetrate);
	Cooccurrencegraph cooccurrencegraph;
	REAL lowerbound;
	REAL upperbound;
	int dimension;
	float cutoff;
	float learningrate;
	float minlearningrate;
};


//-------------------------------------ConceptSpace.cpp------------------------
#include "ConceptSpace.h"

Vector CentralPoint(Vectormap pointslist)
{
	assert(pointslist.size()>0);
	Vectormap::iterator itpoints = pointslist.begin();
	int dimension = (itpoints->second).Size();

	Vector result(dimension, 0);

	for(; itpoints != pointslist.end(); ++itpoints)
	{
		assert((itpoints->second).Size() == dimension);
		for(int i = 0; i < dimension; ++i)
			result[i]+=(itpoints->second)[i];
	}

	for(int i = 0; i < dimension; ++i)
		result[i]/=pointslist.size();

	return result;
}

list<long> ListofFirst(ValueVector input)
{
	list<long> result;
	ValueVector::iterator it;
	for(it=input.begin(); it!= input.end(); it++)
	{
		result.push_back(it->first);
	}
	return result;
}

ConceptSpace::ConceptSpace(int pdimension)
{
	assert(pdimension>0);
	dimension = pdimension;
```

```
}


void ConceptSpace::Initialize(SimpleVectorMap articleset)
// NB Assumption fingerprints are ordered on the basis of weights
{
	int numedges = 0;
	int numnodes = 0;

	cooccurrencegraph.Clear();
	SimpleVectorMap::iterator itfingerprint;
	ValueVector::iterator itconcept1;
	ValueVector::iterator itconcept2;
	srand( (unsigned)time( NULL ) );

	bool stop, stop2;
	Vector randomvector(dimension);
	REAL range = ((REAL)RAND_MAX)*(upperbound-lowerbound);
	int i;
	itfingerprint = articleset.begin();
	for(itfingerprint = articleset.begin(); itfingerprint != articleset.end(); ++itfingerprint)
	{

		stop = false; // Stops the iteration once the critical cutoff was passed

		// Insert the vertices
		for(itconcept1 = (itfingerprint->second).begin(); !stop && itconcept1 != (itfingerprint-
>second).end(); ++itconcept1)
		{
			if(itconcept1->second > cutoff)
			{
				if(!cooccurrencegraph.ContainsVertex(itconcept1->first))
				{
					for(i = 0; i < dimension; ++i)
					{
						randomvector[i] = ((REAL)rand())/range + lowerbound;
					}
					numnodes++;

	cooccurrencegraph.InsertVertex(IDVertex<long,Vector>(itconcept1->first, randomvector));
				}
			}
			else
				stop = true;
		}

		// Insert the edges
		stop = false;
		stop2 = false;
		int abc;
		int def;
		for(itconcept1 = (itfingerprint->second).begin(); !stop && itconcept1 != (itfingerprint-
>second).end(); ++itconcept1)
		{
			itconcept2 = itconcept1;
			abc = itconcept1->first;
			++itconcept2;

			if(itconcept2->second > cutoff)
			{
```

```
                                        stop2 = false;
                                        for(; !stop2 && itconcept2 != (itfingerprint->second).end(); ++itconcept2)
                                        {
                                                if(itconcept2->second > cutoff)
                                                {
                                                        def = itconcept2->first;
                                                        if(!cooccurrencegraph.ContainsEdge(itconcept1->first,
itconcept2->first))
                                                        {
                                                                cooccurrencegraph.InsertEdge(itconcept1-
>first,itconcept2->first);
                                                                numedges++;
                                                        }
                                                        // Add the article number
                                                        cooccurrencegraph.AddEdgeAttribute(itconcept1-
>first,itconcept2->first, itfingerprint->first);
                                                        cooccurrencegraph.AddEdgeAttribute(itconcept2-
>first,itconcept1->first, itfingerprint->first);
                                                }
                                                else
                                                        stop2 = true;
                                        }
                                }
                                else
                                        stop = true;
                        }

        }
        cout << "Edges : " << numedges << '\n';
        cout << "Nodes : " << numnodes << '\n';
}

REAL ConceptSpace::Distance(long a, long b)
{
        assert(cooccurrencegraph.ContainsVertex(a)&&cooccurrencegraph.ContainsVertex(b));
        return (cooccurrencegraph.Retrieve(a)-cooccurrencegraph.Retrieve(b)).L2();
}

REAL ConceptSpace::Attract(const Vector& pcentralpoint, long id, float learningrate)
{
        Vector vec = cooccurrencegraph.Retrieve(id);
        Vector dif = pcentralpoint - vec;
        dif = learningrate*dif/dif.L2();
        vec = vec + dif;
        cooccurrencegraph.Update(vec, id);
        return dif.L2();
}

REAL ConceptSpace::LearnArticle(ValueVector fingerprint)
// NB Assumption fingerprints are ordered on the basis of weights
{
        REAL change = 0;

        ValueVector::iterator itconcept = fingerprint.begin();
        Vectormap relconcepts;

        bool stop = false; // Stops the iteration once the critical cutoff was passed
        for(itconcept; !stop && itconcept != fingerprint.end(); ++itconcept)
        {
                if(itconcept->second > cutoff)
```

```
		{
			Vectormap::value_type temp(itconcept->first,cooccurrencegraph.Retrieve(itconcept-
>first));
			relconcepts.insert(temp);
		}
		else
			stop = true;
	}
	Vector attractor = CentralPoint(relconcepts);



	// learn articles
	for(Vectormap::iterator itattracted = relconcepts.begin(); itattracted != relconcepts.end(); ++itattracted)
	{
		change += Attract(attractor,itattracted->first, 2);
	}

	// forget the rest
	Vectormap space = cooccurrencegraph.Vertices();

	for(Vectormap::iterator itnonrel = space.begin(); itnonrel != space.end(); ++itnonrel)
	{
		if(relconcepts.find(itnonrel->first) == relconcepts.end())
			change += Deter(attractor,itnonrel->first, 1);
	}

	return change;
}


void ConceptSpace::Train(SimpleVectorMap articleset, int numepochs)
{
	assert(numepochs >= 0);


	learningrate = 2/((numepochs+1)<minlearningrate ? (numepochs+1):minlearningrate);
	for(int epoch = 0; epoch < numepochs; ++epoch)
	{
		REAL change = 0;
		learningrate = 2/((epoch+1)<minlearningrate ? (epoch+1):minlearningrate);
		SimpleVectorMap::iterator itarticle;


		for(itarticle = articleset.begin(); itarticle != articleset.end(); ++itarticle)
		{
			change += LearnArticle(itarticle->second);
//			cout << "change " << change << '\n';
		}
		cout << "\nEpoch " << epoch << " total change\t\t" << change;


	}
}

inline REAL ConceptSpace::Deter(const Vector& pcentralpoint, long id, float threshold)
{
	Vector vec = cooccurrencegraph.Retrieve(id);
	Vector dif = pcentralpoint - vec;
	REAL delta = (dif.L2());
```

```
		REAL labda = (delta < threshold)? threshold : (threshold/delta);
		dif = labda * dif/delta;
		vec = vec - dif;
		cooccurrencegraph.Update(vec, id);
		return dif.L2();
}

SpanBox ConceptSpace::ClusterBox(list<long> conceptlist)
{
		SpanBox result;
		list<long>::iterator it;
		for(it = conceptlist.begin(); it != conceptlist.end(); it++)
		{
				result.Insert(cooccurrencegraph.Retrieve(*it));
		}
		return result;
}
```

BIBLIOGRAPHY

1. Wyatt J. Use and sources of medical knowledge. Lancet 1991;338(8779):1368-73.

2. Savitha, Susan. Why evidence-based oncology? Evidence-based Oncology 1999;0(0).

3. Sackett DL, Haynes RB, Guyatt GH, Tugwell P. Clinical Epidemiology.
A Basic Science for Clinical Medicine, 2nd ed. Boston: Little, Brown and Co; 1991.

4. Swanson DR, Smalheiser NR. An interactive system for finding complementary literatures: a stimulus to scientific discovery. Artificial Intelligence 1997;91:183-203.

5. Blois MS. Information and medicine; The nature of medical descriptions. Berkely: University of California Press; 1984.

6. UMLS knowledge sources; Documentation. 6th experimental edition ed: National Library of Medicine; 1995.

7. Schuemie M. Associatieve Conceptuele Ruimte, een vorm van kennisrepresentatie ten behoeve van informatie-zoeksystemen [Master Thesis]. Rotterdam, The Netherlands: Erasmus; 1998.

8. Campbell KE, Oliver DE, Spackman KA, Shortliffe EH. Representing thoughts, words, and things in the UMLS. J Am Med Inform Assoc 1998;5(5):421-31.

9. Sowa JF. Ontology, Metadata, and Semiotics. In: Ganter B, Mineau GW, editors. Conceptual Structures: Logical, Linguistics, and Computational Issues; Lecture Notes in AI#1867. Berlin: Springer-Verlag; 2000. p. 55-81.

10. PubMed Overview. In.; 2001.

11. Craven M, Kumlien J. Constructing biological knowledge bases by extracting information from text sources. Proc Int Conf Intell Syst Mol Biol 1999:77-86.

12. Yakushiji A, Tateisi Y, Miyao Y, Tsujii J. Event extraction from biomedical papers using a full parser. Pac Symp Biocomput 2001:408-19.

13. Sekimizu T, Park HS, Tsujii J. Identifying the Interaction between Genes and Gene Products Based on Frequently Seen Verbs in Medline Abstracts. Genome Inform Ser Workshop Genome Inform 1998;9:62-71.

14. Rindflesch TC, Tanabe L, Weinstein JN, Hunter L. EDGAR: extraction of drugs, genes and relations from the biomedical literature. Pac Symp Biocomput 2000:517-28.

15. Weeber M. Literature-based Discovery in Biomedicine. Groningen: Rijksuniversiteit Groningen; 2001.

16. Stapley BJ, Benoit G. Biobibliometrics: information retrieval and visualization from co- occurrences of gene names in Medline abstracts. Pac Symp Biocomput 2000:529-40.

17. Veeling A, Van der Weerd P. Conceptual grouping in word co-occurrence networks. In: Proceedings of IJCAI '99; 1999: Morgan Kaufmann Publishers, San Francisco, USA; 1999. p. 694-699.

18. Sparck Jones K. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 1972(28):11-20.

19. Decker R, Hirshfield S. Working classes; data structures and algorithms using C++. Boston, USA: PWS Publishing Company; 1996.

20. Nature Genetics. London: Nature Publishing Group.

21. Weiss MA. Data structures and algorithm analysis in C++. 2nd ed. Reading, Massachusetts: Addison Wesly Longman, Inc.; 1999.

22. Cherkassy BV, Goldberg AV, Radzik T. Shortest ath algorithms: Theory and experimental evaluations. In: ACM-SIAM Symposium on Discrete Algortihms (A conference on Theoretical and Experimental Analysis of Discrete Algorithms; 1993; 1993.

23. Bom MFM. Het bepalen van afstanden met het basisnetwerk. Rotterdam: Erasmus Universiteit; 1986.

24. Hebb DO. The Organization of Behavior: A Neuropsychological Theory: John Wiley & Sons, Inc.; 1966.

25. Kohonen T. Self-organizing maps. Berlin: Springer Verlag; 1995.

26. Mulder HM. Discrete Wiskunde. Rotterdam: Econometrisch Instituut, Erasmus Universiteit; 1999.